



REAL TIME OBJECT DETECTION USING RASPBERRY PI

A PROJECT REPORT

Submitted by

P.ARUN KARTHIK

311516105008

S.BALAJI

311516105013

T.R.GURUPRASAADH

311516105017

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

in

ELECTRICAL AND ELECTRONICS ENGINEERING

MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE

KODAMBAKKAM, CHENNAI-600 024

ANNA UNIVERSITY: CHENNAI- 600 025

APRIL 2020

BONAFIDE CERTIFICATE

This is to certify that the project report on “**REAL TIME OBJECT DETECTION USING RASPBERRY PI**” is the authenticated work of **P. ARUN KARTHIK, S.BALAJI and T.R.GURUPRASAADH** who worked on the project under our guidance.

SIGNATURE

Mrs. S.SOUNDARABALA,M.E.,

HEAD OF THE DEPARTMENT

Department of Electrical and
Electronics Engineering,
Meenakshi Sundararajan
Engineering College,
363, Arcot Road
Chennai- 600024

SIGNATURE

Mrs. S.SOUNDARABALA,M.E.,

INTERNAL GUIDE

Department of Electrical and
Electronics Engineering,
Meenakshi Sundararajan
Engineering College,
363, Arcot Road
Chennai- 600024

Submitted for the VIVA VOCE held on

INTERNAL EXAMINER

EXTERNAL EXAMINER

Vivegam Technology Solutions
No3, Sai Complex, 2nd Floor,
Staion View Road,
Kodambakkam, Chennai – 600 024.
Ph : +91-44-42045388
Email : sales@vivegamtech.com



March 24th, 2020.

TO WHOMSOEVER IT MAY CONCERNS

This is to certify that **Mr. T.R.Guruprasaadh** studying Final Year B.E Electrical and Electronics Engineering at Meenakshi Sundararajan Engineering College, kodambakkam, Chennai-600024 has successfully completed the final year project " **Real Time Object Detection using Raspberry Pi**" in our organisation from 18/12/2019 to 24/03/2020.

His performance during the project work was excellent.

We wish him all the best in his future endeavours.

For Vivegam Technology Solutions

A handwritten signature in black ink, appearing to be "C. Hariharan", is written over a horizontal line.

(C.Hariharan)

Vivegam Technology Solutions
No3, Sai Complex, 2nd Floor,
Staion View Road,
Kodambakkam, Chennai – 600 024.
Ph : +91-44-42045388
Email : sales@vivegamtech.com



March 24th, 2020.

TO WHOMSOEVER IT MAY CONCERNS

This is to certify that **Mr. S.Balaji** studying Final Year B.E Electrical and Electronics Engineering at Meenakshi Sundararajan Engineering College, kodambakkam, Chennai-600024 has successfully completed the final year project " **Real Time Object Detection using Raspberry Pi**" in our organisation from 18/12/2019 to 24/03/2020.

His performance during the project work was excellent.

We wish him all the best in his future endeavours.

For Vivegam Technology Solutions

A handwritten signature in black ink, appearing to be "C. Hari Haran", is written over the text "For Vivegam Technology Solutions".

(C.Hari Haran)

March 24th, 2020.


TO WHOMSOEVER IT MAY CONCERNS

This is to certify that **Mr. P. Arun Karthik** studying Final Year B.E Electrical and Electronics Engineering at Meenakshi Sundararajan Engineering College, kodambakkam, Chennai-600024 has successfully completed the final year project "**Real Time Object Detection using Raspberry Pi**" in our organisation from 18/12/2019 to 24/03/2020.

His performance during the project work was excellent.

We wish him all the best in his future endeavours.

For Vivegam Technology Solutions


(C.Hari Haran)

ACKNOWLEDGEMENT

This project experience has been a rewarding one, not only for having opened up new vistas of knowledge but also for kindling the spirit of team work.

We wish to place on record our deep sense of gratitude to our Honorable Secretary, **Dr.K.S.Babai, B.Sc., M.S., Ph.D, FIE, Meenakshi Sundararajan Engineering College, Kodambakkam, Chennai-600 024** for her constant encouragement with a well-equipped laboratories and library to get our knowledge enhanced in our project.

We express our heartfelt thanks to our Head of the Department and our Internal Guide **Mrs.S.SowndaraBala, Meenakshi Sundararajan Engineering College, Kodambakkam, Chennai-600 024** whose guidance and support have contributed in a large measure in developing and completion of our project work. We consider it a privilege to have been a student under her.

We like to thank the members of the review committee and their valuable suggestion with constructive criticisms on several occasions that enabled us to mould our project successfully.

TABLE OF CONTENTS

S.NO.	DESCRIPTION	PAGE NO.
	ABSTRACT	ix
	LIST OF FIGURES	x
	LIST OF ABBREVIATIONS	xiv
1	INTRODUCTION	1
	1.1 OBJECT DETECTION	2
	1.1.1 METHODS OF OBJECT DETECTION	2
	1.2 ARTIFICIAL INTELLIGENCE VS MACHINE LEARNING VS DEEP LEARNING	4
	1.2.1 DEEP LEARNING	4
	1.2.1.1 WORKING OF DEEP LEARNING MODEL	5
	1.3 NEURAL NETWORKS	5
	1.3.1 TYPES OF NEURAL NETWORKS	6
	1.3.1.1 ARTIFICIAL NEURAL NETWORKS	7
	1.3.1.2 DEEP NEURAL NETWORK	7
	1.3.1.3 CONVOLUTION NEURAL NETWORK	9
	1.4 TRANSFER LEARNING	10
	1.4.1 WHY USE TRANSFERLEARNING	10
	1.5 PRETRAINED MODELS	12
	1.6 EFFICIENT CONVOLUTIONAL NEURAL NETWORK	13
	1.6.1 CONVOLUTIONAL NEURAL NETWORK	13
	1.7 CONVOLUTION	14

	1.7.1 GROUPED CONVOLUTION	15
	1.7.2 DEPTHWISE CONVOLUTION	16
	1.8 MOBILE NET – V2	17
2	LITERATURE SURVEY AND BLOCK DIAGRAM	18
	2.1 LITERATURE SURVEY	18
	2.2 PROPOSED MODEL	20
	2.3 OVERVIEW OF PROJECT	21
3	HARDWARE COMPONENTS	
	3.1 GENERAL DESCRIPTION	23
	3.1.1 PI CAMERA MODULE	23
	3.1.2 HARDWARE	24
	3.1.3 ADDITIONAL HARDWARE SPECIFICATIONS	26
	3.2 PIN DIAGRAM OF RASPBERRY PI 3 B+	27
	3.3 WEBCAM	29
	3.3.1 INTERFACE	30
	3.3.2 USES	31
	3.4 RASPBERRY PI CAMERA	31
	3.4.1 SPECIFICATIONS	32
	3.4.2 APPLICATIONS	32
4	SOFTWARE REQUIREMENT	33
	4.1 GENERAL	33
	4.1.1 VNC VIEWER	33
	4.1.1.1 USES OF VNC	34
	4.1.2 PuTTY	34
	4.1.2.1 PURPOSE OF USING PuTTY	35

4.1.2.1.1 USED AS A FILE TRANFER PROTOCOL	35
4.1.2.1.2 USED TO GENERATE HASH KEY	35
4.1.3 OPENCV	35
4.1.4 OPENCV PYTHON	36
4.1.5 TENSORFLOW LITE	37
4.1.6 RASPBIAN OS	37
4.2 INSTALLATION OF RASPBIAN OPERATING SYSTEMRASPBERRY PI	38
4.3 CAMERA MODULE CONFIGURATION IN RASPBERRYPI	43
4.3.1 TO CAPTURE A IMAGE WITH RASPBERRY PI CAMERA MODULE	46
4.3.2 TO RECORD A VEDIO WITH RASPBERRY PI CAMERAMODULE	46
5 TESTING AND TRAINING	
5.1 TENSOR FLOW	47
5.1.1 HISTORY OF TENSOR FLOW	47
5.1.2 TENSORFLOW ARCHITECTURE	47
5.1.3 RUNNING TENSORFLOW	48
5.1.4 COMPONENTS OF TENSORFLOW	48
5.2 TRAINING TENSORFLOW OBJECT DETECTION CLASSIFIER	49
5.2.1 ANACONDA	49
5.2.1.1 DOWNLOAD ANACONDA	50
5.2.1.2 INSTALL ANOCONDA	50
5.2.2.1 SETTING UP THE OBJECT DETECTION DIRECTORY STRUCTURE AND ANACONDA VIRTUAL ENVIRONMENT	51

5.2.2. 1 TENSORFLOW OBJECT DETECTION API	51
5.2.2.2 SSD MOBILENET V2 COCO MODEL FROM TENSORFLOW'S MODEL ZOO	52
5.2.2.3 ANOCONDA VIRTUAL ENVIRONMENT	53
5.2.2.4 TENSORFLOW INSTALLATION	53
5.2.2.5 CONFIGURING PYTHON PATH ENVIRONMENTVARIABLE	54
5.2.3 GATHERING AND LABEELING IMAGES	54
5.2.3.1 GATHER IMAGES	54
5.2.3.2 LABEL PICTURES	55
5.2.4 GATHERING TRAINING DATA	57
5.2.4.1 CSV FILE	58
5.2.4.2 TFRECORD	58
5.2.4.2.1 STRUCTURING TFRECORDS	59
5.2.5 CREATING LABEL MAP AND CONFIGURE TRAINING	60
5.2.5.1 LABEL MAP	60
5.2.5.2 CONFIGURE TRAINING	60
5.2.6 RUN THE TRAINING	62
5.2.7 EXPORT INFERENCE GRAPH	65
5.3 TENSORFLOW LITE OBJECT DETECTION	67
5.3.1 INTRODUCTION	67
5.3.1.1 FEATURES	67
5.3.1.2 COMPONENTS	68
5.3.1.3 ARCHITECTURE	69
5.3.2 EXPORT FROZEN INFERENCE GRAPH FOR TENSORFLOW LITE	69
5.3.3 CREATING OPTIMIZED TENSORFLOW LITE MODEL	70

	5.3.3.1 ALGORITHM	71
	5.3.3.2 TFLITE COMMAND	71
	5.3.4 TENSORFLOW OBJECT DETECTION API ON RASPBERRY PI	73
6	CONCLUSION AND FUTURE SCOPE	80
	REFERENCES	81
	APPENDIX	82

ABSTRACT

Object Detection is widely utilized in several applications such as detecting vehicles, face detection, autonomous vehicles and pedestrians on streets. TensorFlow's Object Detection API is a powerful tool that can quickly enable anyone to build and deploy powerful image recognition software. Object detection not solely includes classifying and recognizing objects in an image however additionally localizes those objects and attracts bounding boxes around them. This paper mostly focuses on setting up the environment and tflite model for detecting hotel items like jug, cup and flask. We have got Tensor flow Object Detection API to train model and we have used Single Shot Multibox Detector (SSD) MobileNet V2 algorithm for implementation.

LIST OF FIGURES

FIGURE NO.	DESCRIPTION	PAGE NO.
1.1	AI vs ML vs DL	4
1.2	Basic Architecture of a Neural Network	6
1.3	Basic Architecture of a Deep Neural Networks	8
1.4	Basic Architecture of a Convolutional Neural Networks	10
1.5	Transfer Learning performance curve	11
1.6	COCO model used for PC,RDP	12
1.7	COCO model used for embedded devices	13
1.8	Computation of CNN	13
1.9	3x3 Convolution	14
1.10	1x1 Convolution	15
1.11	Grouped Convolution 3x3 with number groups = 2	15
1.12	Grouped Convolution 3x3 with number groups = 3	15
1.13	Grouped Convolution 1x1 with number groups = 2	16
1.14	Grouped Convolution 1x1 with number groups = 3	16
1.15	Depth wise Convolution	16
1.16	Convolution Mobile NetV2	17

2.1	Block Diagram	21
3.1	Raspberry Pi 3 B+ and Pi Came	25
3.2	Pin diagram of Raspberry Pi 3 B+ Model	27
3.3	Blocks of Raspberry Pi 3 mode	29
3.4	Raspberry pi camera	31
4.1	Stretch Version and its compatibility of different Raspberry Pi's	38
4.2	Downloading Raspbian Stretch	40
4.3	Using Etcher for flashing	41
4.4	Opening Raspberry Pi Configuration	42
4.5	Enabling System in Raspberry Pi Configuration	42
4.6	Enabling SSH and VNC in Raspberry Pi Configuration	43
4.7	Placing Cable in the Raspberry Pi camera port	44
4.8	Selecting Interfacing Option in Configuration Tool	45
4.9	Enabling Pi Camera to establish connection	45
5.1	Download Python 3.6 version of Anaconda	50
5.2	Installing Anaconda 3.5.0.1 version	51
5.3	Created virtual environment and python version	53
5.4	Gathering images for annotation	55

5.5	Labeling images using LabelImg	56
5.6	XML content generated for an image	57
5.7	XML is converted to CSV	58
5.8	Generated csv file	58
5.9	Data content inside a CSV file	58
5.10	Generation of record file	59
5.11	Generated recorded files	59
5.12	Creating label map	60
5.13	Training command	62
5.14	Training process	64
5.15	Command for Tensorboard	64
5.16	Total loss graph	65
5.17	Command used for generating inference graph	66
5.18	Files inside Inference graph folder	66
5.19	Setting up environment variables	69
5.20	Command for exporting TFLite model	69

5.21	Files inside TFlite folder	70
5.22	Command used for Creating detect.tflite file	71
5.23	Generated TFlite_File	72
5.24	Generated detect.tflite file in Raspberry Pi	73
5.25	Issuing update and upgrade commands in terminal window	73
5.26	Setting pythonpath in terminal window	74
5.27	Enabling Camera Module in Raspberry Pi configuration menu	78
5.28	Final Output of testing	79

LIST OF TABLES

TABLE NO.	DESCRIPTION	PAGE NO.
3.1	Raspberry Pi hardware specifications	24
3.2	Pin Specifications of Raspberry Pi 3B+ Model	29
5.1	Versions specified Libraries	53
5.2	Components in TensorFlow Lite	68

LIST OF ABBREVIATION

S.NO	ABBREVIATION	EXPANSION
1.	CNN	Convolutional Neural Network
2.	ANN	Artificial Neural Network
3.	RELU	Rectified Linear Unit
4.	SVM	Support Vector Machine
5.	SSD	Single Shot MultiBox Detector
6.	TF	TensorFlow
7.	TFLite	TensorFlow Lite
8.	RPI	Raspberry PI
9.	VNC	Virtual Network Computing
10.	CSV	Comma-Separated Values
11.	XML	Extensible Markup Language
12.	PIP	Pip Install Package

13.	ML	Machine Learning
14.	GPU	Graphic Processing Unit
15.	RDP	Remote Desktop
16.	SSH	Secure Shell
17.	CV	Computer Vision
18.	SIFT	Scale-Invariant Feature Transform
19.	YOLO	You Look Only Once
20.	CAP	Credit Assignment Path
21.	DNN	Deep Neural Network
22.	TL	Transfer Learning
23.	ARM	Advanced RISC Machine
24.	POE	Power Over Ethernet
25.	API	Application Program Interface
26.	UVC	USB Video Device Class

27.	PIL	Python Imaging Library
28.	FOSS	Free and Open Source Software
29.	RCNN	Region Convolution Neural Network
30.	CKPT	Check Point Process
31.	TOCO	Tensorflow Lite Optimizing Converter
32.	AI	Artificial Intelligence

CHAPTER 1

INTRODUCTION

Computer Vision (CV) is the science of understanding and manipulating digital videos and images. Computer Vision plays a vital role in many applications, which includes Face recognition, image retrieval, industrial inspection, and augmented reality etc. With the emergence of deep learning, computer vision has proven to be useful for various applications. Deep Learning is an Artificial Neural Network (ANN) collection of methods, which is a machine learning branch. On the human brain, ANNs are modelled where nodes are connected to each other that pass data to each other. The use of deep learning for computer vision can be categorized into various categories: generation, segmentation, detection and classification of both videos and images. Image classification labels the image as a whole Finding the position of the object in addition to labelling the object is called object localization. Typically, the position of the object is defined by rectangular coordinates. Finding multiple objects in the image with rectangular coordinates is called detection. Segmentation is detecting exact objects like creating a transparent mask above the object with exact edges. An image classification or model of image recognition merely detects an object's likelihood in an image. In comparison the location of objects relates to the place of an item in the picture. A localization algorithm for object will output the place coordinates of an item with regard to the picture or image. Object detection is a problem of importance in CV. Similar to image classification tasks, deeper networks have shown better performance in detection. At present, the accuracy of these techniques is excellent. Hence it used in many applications. The difference is the number of objects. In detection, there are a variable number of objects. This small difference makes a big difference when designing the architectures for the deep learning model concerning localization or detection.

Let's see in detail about the terminologies involved in our project

1.1 OBJECT DETECTION

Object detection is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos. Well-researched domains of object detection include face detection and pedestrian detection. Object detection has applications in many areas of computer vision, including image retrieval and video surveillance.

It is widely used in computer vision tasks such as image annotation, activity recognition, face detection, face recognition, video object co-segmentation. It is also used in tracking objects, for example tracking a ball during a football match, tracking movement of a cricket bat, or tracking a person in a video.

Every object class has its own special features that helps in classifying the class – for example all circles are round. Object class detection uses these special features. For example, when looking for circles, objects that are at a particular distance from a point (i.e. the center) are sought. Similarly, when looking for squares, objects that are perpendicular at corners and have equal side lengths are needed. A similar approach is used for face identification where eyes, nose, and lips can be found and features like skin color and distance between eyes can be found.

1.1.1 METHODS OF OBJECT DETECTION

Methods for object detection generally fall into either machine learning-based approaches or deep learning-based approaches. For Machine Learning approaches, it becomes necessary to first define features using one of the methods below, then using a technique such as support vector machine (SVM) to do the classification. On the other hand, deep learning techniques are able to do end-to-end object detection without specifically defining features, and are typically based on convolutional neural networks (CNN).

Machine learning approaches:

- Viola–Jones object detection framework based on Haar features
- Scale-invariant feature transform (SIFT)
- Histogram of oriented gradients (HOG) features

Deep learning approaches:

- Region Proposals (R-CNN, Fast R-CNN, Faster R-CNN)
- Single Shot MultiBox Detector (SSD)
- You Only Look Once (YOLO)
- Single-Shot Refinement Neural Network for Object Detection (RefineDet)
- Retina-Net
- Deformable convolutional networks

For this project we have used a deep learning-based technique based on Convolutional Neural Networks (CNN). The approach used is Single Shot MultiBox Detector (SSD). The model used is: SSD_MobileNet_v2_coco_model.

1.2 ARTIFICIAL INTELLIGENCE VS MACHINE LEARNING VS DEEP LEARNING

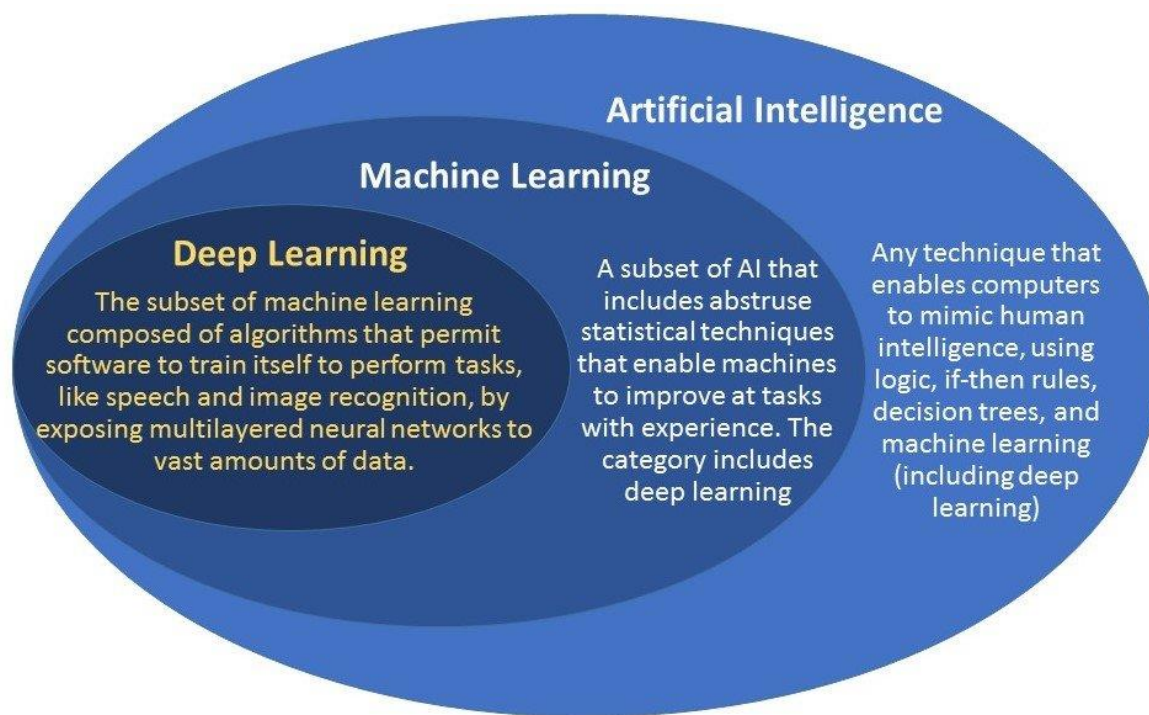


Figure 1.1 AI vs ML vs DL

1.2.1 DEEP LEARNING

Deep is a technical term. It refers to the number of layers in a neural network. A shallow network has one so-called hidden layer, and a deep network has more than one. Multiple hidden layers allow deep neural networks to learn features of the data in a so-called

feature hierarchy, because simple features (e.g. two pixels) recombine from one layer to the next, to form more complex features (e.g. a line).

Nets with many layers pass input data (features) through more mathematical operations than nets with few layers, and are therefore more computationally intensive to train.

Computational intensivity is one of the hallmarks of deep learning, and it is one reason why GPUs are in demand to train deep-learning models.

1.2.1.1 WORKING OF DEEP LEARNING MODEL:

In deep learning, each level learns to transform its input data into a slightly more abstract and composite representation. In an image recognition application, the raw input may be a matrix of pixels; the first representational layer may abstract the pixels and encode edges; the second layer may compose and encode arrangements of edges; the third layer may encode a nose and eyes; and the fourth layer may recognize that the image contains a face.

Importantly, a deep learning process can learn which features to optimally place in which level on its own. The "deep" in "deep learning" refers to the number of layers through which the data is transformed. More precisely, deep learning systems have a substantial credit assignment path (CAP) depth. The CAP is the chain of transformations from input to output. CAPs describe potentially causal connections between input and output. For a feedforward neural network, the depth of the CAPs is that of the network and is the number of hidden layers plus one (as the output layer is also parameterized).

For recurrent neural networks, in which a signal may propagate through a layer more than once, the CAP depth is potentially unlimited. No universally agreed upon threshold

of depth divides shallow learning from deep learning, but most researchers agree that deep learning involves CAP depth > 2 . CAP of depth 2 has been shown to be a universal approximator in the sense that it can emulate any function.] Beyond that more layers do not add to the function approximator ability of the network. Deep models (CAP > 2) are able to extract better features than shallow models and hence, extra layers help in learning features.

1.3 NEURAL NETWORKS:

A neural network is a network or circuit of neurons, or in a modern sense, an artificial neural network, composed of artificial neurons or nodes.[1] Thus a neural network is either a biological neural network, made up of real biological neurons, or an artificial neural network, for solving AI problems.

The connections of the biological neuron are modeled as weights. A positive weight reflects an excitatory connection, while negative values mean inhibitory connections. All inputs are modified by a weight and summed.

This activity is referred as a linear combination. Finally, an activation function controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1, or it could be -1 and 1 .

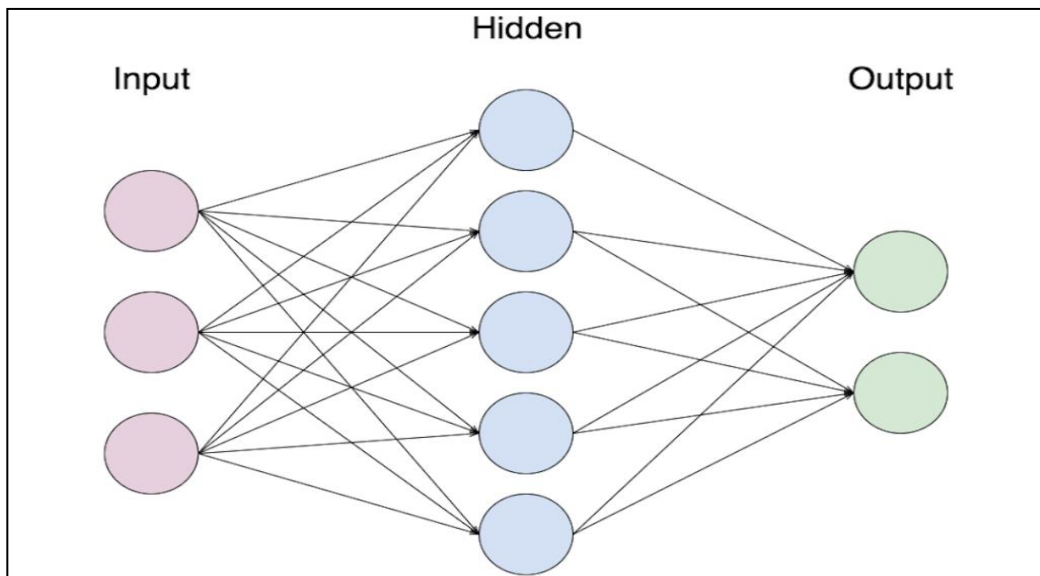


Figure 1.2 Basic Architecture of a Neural Network

1.3.1 TYPES OF NEURAL NETWORK:

- 1) Artificial Neural networks
- 2) Deep Neural networks
- 3) Convolutional Neural networks

1.3.1.1 ARTIFICIAL NEURAL NETWORKS:

An Artificial Neural Network ANN is based on a collection of connected units called artificial neurons, (analogous to biological neurons in a biological brain). Each connection (synapse) between neurons can transmit a signal to another neuron. The receiving (postsynaptic) neuron can process the signal(s) and then signal downstream neurons connected to it. Neurons may have state, generally represented by real numbers, typically between 0 and 1. Neurons and synapses may also have a weight that varies as learning proceeds, which can increase or decrease the strength of the signal that it sends downstream. Typically, neurons are organized in layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first(input), to the last (output) layer, possibly after traversing the layers multiple times.

The original goal of the neural network approach was to solve problems in the same way that a human brain would. Over time, attention focused on matching specific mental abilities, leading to deviations from biology such as backpropagation, or passing information in the reverse direction and adjusting the network to reflect that information. Neural networks have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and video games and medical diagnosis.

As of now neural networks typically have a few thousand to a few million units and millions of connections. Despite this number being several orders of magnitude less than the number of neurons on a human brain, these networks can perform many tasks at a level beyond that of humans.

1.3.1.2 DEEP NEURAL NETWORK:

A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers. The DNN finds the correct mathematical manipulation to turn the input into the output, whether it be a linear relationship or a non-linear relationship. The network moves through the layers calculating the probability of each output. For example, a DNN that is trained to recognize dog breeds will go over the given image and calculates the probability that the dog in the image is a certain breed. The user can review the results and select which probabilities the network should display (above a certain threshold, etc.) and return the proposed label. Each mathematical manipulation as such is considered a layer, and complex DNN have many layers, hence the name "deep" networks. The goal is that eventually, the network will be trained to decompose an image into features, identify trends that exist across all samples and classify new images by their similarities without requiring human input. DNNs can model complex non-linear relationships. DNNs are typically feedforward networks in which data flows from the input layer to the output layer without looping back.

At first, the DNN creates a map of virtual neurons and assigns random numerical values, or "weights", to connections between them. The weights and inputs are multiplied and return an output between 0 and 1. If the network didn't accurately recognize a particular pattern, an algorithm would adjust the weights. That way the algorithm can make certain parameters more influential, until it determines the correct mathematical manipulation to fully process the data.

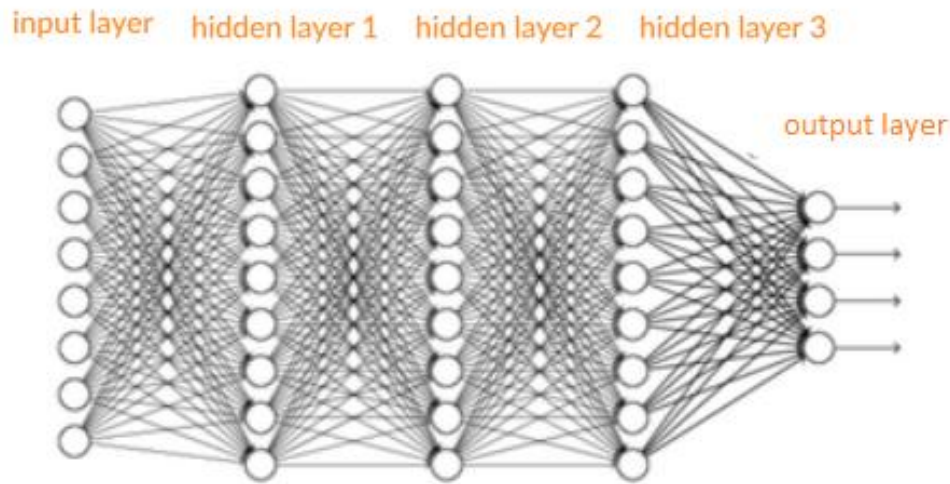


Figure 1.3 Architecture of Deep Neural Networks

1.3.1.3 CONVOLUTIONAL NEURAL NETWORKS:

A convolutional network receives a normal color image as a rectangular box whose width and height are measured by the number of pixels along those dimensions, and whose depth is three layers deep, one for each letter in RGB. Those depth layers are referred to as channels.

As images move through a convolutional network, we will describe them in terms of input and output volumes, expressing them mathematically as matrices of multiple dimensions in this form: $30 \times 30 \times 3$. From layer to layer, their dimensions change for reasons that will be explained below. Now, for each pixel of an image, the intensity of R, G and B will be expressed by a number, and that number will be an element in one of the three, stacked two-dimensional matrices, which together form the image volume. Those numbers are the initial, raw, sensory features being fed into the convolutional network, and the ConvNets purpose is to find which of those numbers are significant signals that actually help it classify images more accurately. (Just like other feed forward networks we have discussed.)

Rather than focus on one pixel at a time, a convolutional net takes in square patches of pixels and passes them through a filter. That filter is also a square matrix smaller than the image itself, and equal in size to the patch. It is also called a kernel, which will ring a bell for those familiar with support-vector machines, and the job of the filter is to find patterns in the pixels.

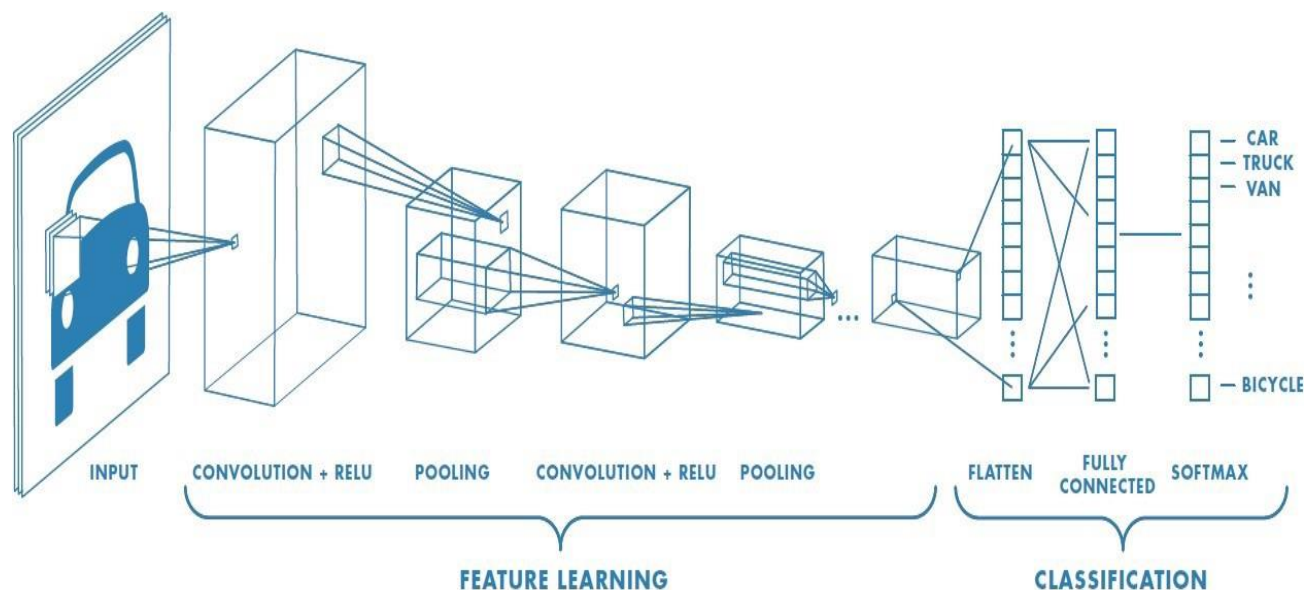


Figure 1.4 Architecture of Convolutional Neural Network

1.4 TRANSFER LEARNING

Transfer learning (TL) is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks. This area of research bears some relation to the long history of psychological literature on transfer of learning, although formal ties between the two fields are limited. From the practical standpoint, reusing or transferring information from previously learned tasks for the learning of new tasks has the potential to significantly improve the sample efficiency of a reinforcement learning agent.

1.4.1 WHY USE TRANSFER LEARNING?

Transfer learning is an optimization, a shortcut to saving time or getting better performance. In general, it is not obvious that there will be a benefit to using transfer learning in the domain until after the model has been developed and evaluated.

The three possible benefits to look for when using transfer learning:

- **Higher start.** The initial skill (before refining the model) on the source model is higher than it otherwise would be.
- **Higher slope.** The rate of improvement of skill during training of the source model is steeper than it otherwise would be.
- **Higher asymptote.** The converged skill of the trained model is better than it otherwise would be.

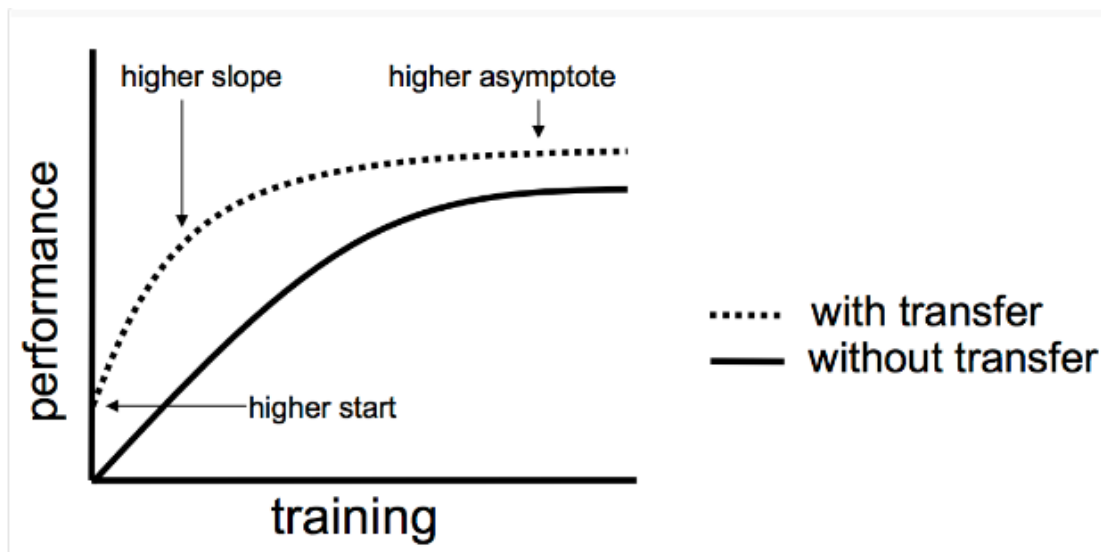


Figure 1.5 Transfer Learning performance curve

1.5 PRETRAINED MODELS

These models can be used on Laptops, PC's.

COCO-trained models

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v1_0.75_depth_coco ☆	26	18	Boxes
ssd_mobilenet_v1_quantized_coco ☆	29	18	Boxes
ssd_mobilenet_v1_0.75_depth_quantized_coco ☆	29	16	Boxes
ssd_mobilenet_v1_ppn_coco ☆	26	20	Boxes
ssd_mobilenet_v1_fpn_coco ☆	56	32	Boxes
ssd_resnet_50_fpn_coco ☆	76	35	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes

faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241		Boxes
faster_rcnn_nas	1833	43	Boxes
faster_rcnn_nas_lowproposals_coco	540		Boxes
mask_rcnn_inception_resnet_v2_atrous_coco	771	36	Masks
mask_rcnn_inception_v2_coco	79	25	Masks
mask_rcnn_resnet101_atrous_coco	470	33	Masks
mask_rcnn_resnet50_atrous_coco	343	29	Masks

Figure 1.6 COCO trained model for PC, RDP

These can be used on android/ ios devices and mobile computing devices like raspberry pi

Mobile models

Model name	Pixel 1 Latency (ms)	COCO mAP	Outputs
ssd_mobilenet_v3_large_coco	119	22.6	Boxes
ssd_mobilenet_v3_small_coco	43	15.4	Boxes

Pixel4 Edge TPU models

Model name	Pixel 4 Edge TPU Latency (ms)	COCO mAP	Outputs
ssd_mobilenet_edgetpu_coco	6.6	24.3	Boxes

Activate Windows
Go to Settings to activate Windows

Figure 1.7 COCO trained model for embedded devices

1.6 EFFICIENT CONVOLUTIONAL NEURAL NETWORK:

1.6.1. CONVOLUTIONAL NEURAL NETWORK:

Before efficient CNN models, the computational cost of building blocks used in efficient CNN models, and performing convolution in spatial and channel domain is explained.

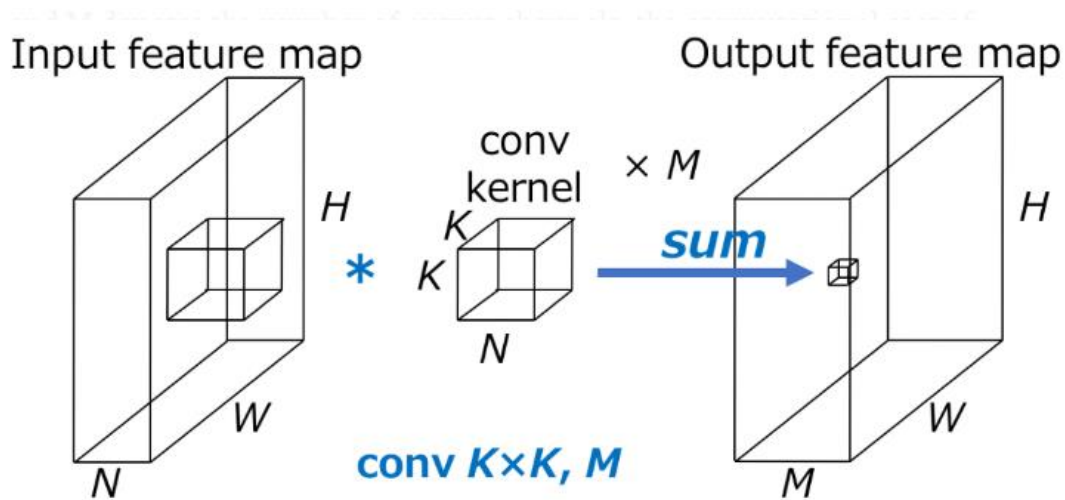


Figure 1.8 Computation of CNN

- $H \times W$ denotes the spatial size of output feature map,
- N denotes the number of input channels
- $K \times K$ denotes the size of convolutional kernel
- M denotes the number of output channels

The computational cost of standard convolution becomes $HWNK^2M$ and it is proportional to

- The spatial size of the output feature map $H \times W$
- The size of convolutional kernel K^2
- The number of input and output channels $N \times M$

The above computational cost is required when convolution is performed on both spatial and channel domain. CNNs can be speeded up by factorizing this convolution.

1.7. CONVOLUTION:

By connecting lines between input and output to visualize dependency between input and output. The number of lines roughly indicate the computational cost of convolution in spatial and channel domain respectively.

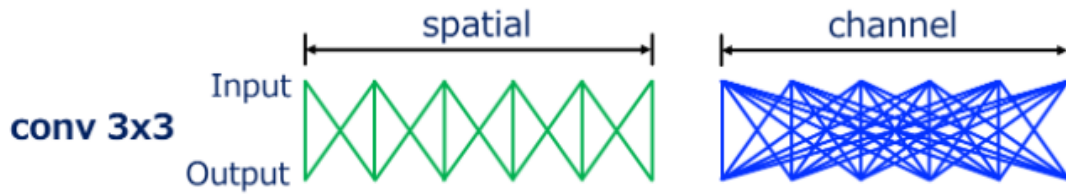


Figure 1.9 3x3 Convolution

In conv3x3, the most commonly used convolution, can be visualized as shown. We can see that the input and output are locally connected in spatial domain while in channel domain, they are fully connected.



Figure 1.10 1x1 Convolution

Next, conv1x1 or pointwise convolution, which is used to change the size of channels. The computational cost of this convolution is $HWNM$ because the size of kernel is 1x1, resulting in 1/9 reduction in computational cost compared with conv3x3, this convolution is used in order to blend information among channels.

1.7.1 GROUPED CONVOLUTION:

Grouped convolution is a variant of convolution where the channels of the input feature map are grouped and convolution is performed independently for each grouped channel. Let G denote the number of groups, the computational cost of grouped convolution is $\frac{HWNK^2M}{G}$, resulting in $\frac{1}{G}$ reduction in computational cost compared with standard convolution.

- The grouped conv3x3 with $G=2$. We can see that the number of connections in channel domain becomes smaller than standard convolution, which indicates smaller computational cost.



Figure 1.11 Grouped Convolution 3x3, with number of groups = 2

- The grouped conv3x3 with $G=3$. The connections become sparser.



Figure 1.12 Grouped Convolution 3x3, with number of groups = 3

- The grouped conv1x1 with $G=2$. Thus, conv1x1 can also be grouped.



Figure 1.13 Grouped Convolution 1x1, with number of groups = 2

- The grouped conv1x1 with $G=3$.



Figure 1.14 Grouped Convolution 1x1, with number of groups = 3

1.7.2 DEPTHWISE CONVOLUTION:

In depth wise convolution, convolution is performed independently for each of input channels. It can be defined as a special case of grouped convolution where the numbers of input and output channels are same and G equals the number of channels. It reduces the computational cost by omitting convolution in channel domain



Figure 1.15 Depthwise Convolution

1.8. MOBILENET-V2:

MobileNet-V2 is composed of an architecture conv1x1, conv3x3, and conv1x1. The first conv1x1 increases the dimensions of the input channel, then depth wise convolution performed. The final conv1x1 decrease the dimension of the output channels.

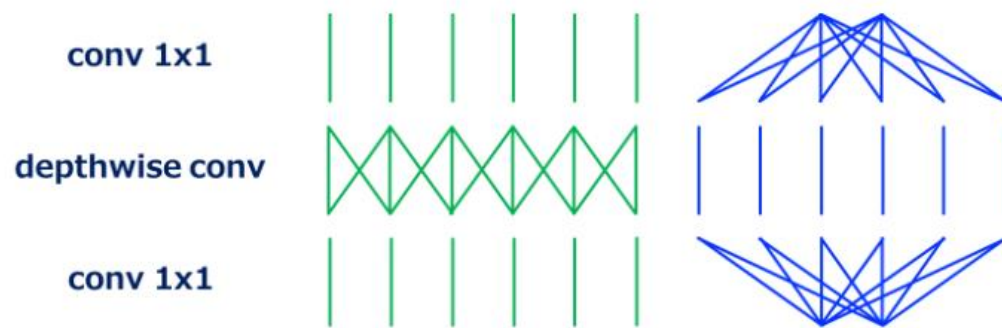


Figure 1.16 Convolution of MobileNet-V2

CHAPTER-2

LITERATURE SURVEY AND BLOCK DIAGRAM

2.1 LITERATURE SURVEY

1. Gayathri *et al* (2019) has proposed a faster model of object detection namely Single Shot Multi-Box Detection (SSD) along with Mobile Net, published in second International Conference on Computational Intelligence in Data Science (ICCIDS-2019)

It is shown that for real time applications where inference needs to be quick, this model does the work elegantly. The paper compares other deep learning models like RCNN, Fast RCNN, Faster RCNN, but owing to their complex networked structure, the accuracy is lesser when detecting multiple objects in a single frame. As SSD uses convolution filters for detecting the objects depth it loses its accuracy if the frame is of low resolution. So we use the Mobile Net technique as it uses depth wise separable convolution which significantly reduces the parameters size when compared with normal convolution filters of same depth. Thus, we can get the light weight deep neural network as a result. On whole SSD with MobileNet is nothing but a model in which meta architecture is SSD and the feature extraction type is MobileNet.

2. Abdelhakim Zeroual (2019) *et al* published in the Proceedings of the International Conference on Networking and Advanced Systems (ICNAS) IEEE and proposed that there are many deep neural network frameworks for embedded platforms. Caffe2 is a framework, which supports multiple embedded platforms such as, Android, iOS, Tegra, and Raspbian. TensorFlow Lite is another is a lightweight and accessible framework and a new trend, which supports a variety of embedded platforms. We can implement and

run deep neural network using Python and C++. In the domain of security on Mobile Cloud Environment (MCC), for facial recognition based on Deep convolutional neural network to authenticate the device, the training and recognition phases were done on cloud because of huge computation. Based on this work, we propose to use the TensorFlow lite for mobile device to reduce the time of transfer between cloud and mobile. By focusing on the cited work, the authors have achieved an accuracy of 100 % compared to the last one with 99.50 %. To reduce the transfer time, they have suggested using a TensorFlow lite framework destined for mobile to do a recognition without a need of cloud, contrary to the previous work.

3. Tiagrajah V (2019) *et al* published in the Proceedings of the 9th International Conference on System Engineering and Technology (ICSET) IEEE have used Convolutional Neural Networks for object detection. They have presented a deep learning approach for robust detection of traffic light by comparing two object detection models and by evaluating the flexibility of the TensorFlow Object Detection Framework to solve the real-time problems. The models are Single Shot Multibox Detector (SSD) MobileNet V2 and Faster-RCNN. They showed that Faster-RCNN delivers 97.015%, which outperforms SSD by 38.806% for a model which had been trained using 441 images. They concluded that SSD shows more false negatives in object detection, but when used in mobile computing devices like Raspberry Pi, Android Devices, the inference time is faster in SSD compared to the Faster RCNN model.

4. Sumeet Sanjay Walam (2018) *et al* published in the Proceedings of the 2nd International Conference on Inventive Communication and Computational Technologies (ICICCT 2018) IEEE, proposes to detect and separate objects from a set based on their color. Proposed method of categorization is done on the basis of color of the object using

a raspberry Pi 3. System categorizes the object into three different colors. The detection of the particular color is done by a light intensity camera to frequency convertor method.

This was the motivation for our mini project; we tried implementing the same on an Arduino board and finally ended with an artificial intelligence project. We used SSD mobileNet model to detect objects. We were able to process the entire computation inside this raspberry pi without having to send it to the cloud.

2.2 PROPOSED MODEL

Based on the papers we came to conclusion to use Raspberry Pi 3B+ for object detection with using SSD-mobilenet V2 model along with TensorFlow lite as the framework. It was decided based on the following reasons:

- Object detection without deep learning algorithms, would mean thousands of lines of code which would be a cumbersome process. With lot of research going under Object Detection and lots of models becoming open-source, there will always be newer models in the future having better accuracy, lesser time for detection. But that is technological advancements, which needs to be dealt in a positive manner.
- A micro-controller like device with onsite computing capabilities will reduce the cost and time of the process by reducing the server latency time and also the need for employing servers for the same.
- An SSD-mobilenet based model gives better accuracy and lesser inference time compared to the other models. Based on the trade-off between time and accuracy, it is safer to select this model.

In our project we try to detect flask, jug and cup and based on the detection result a message is sent to the concerned room-service staff.

2.3 Overview of Project

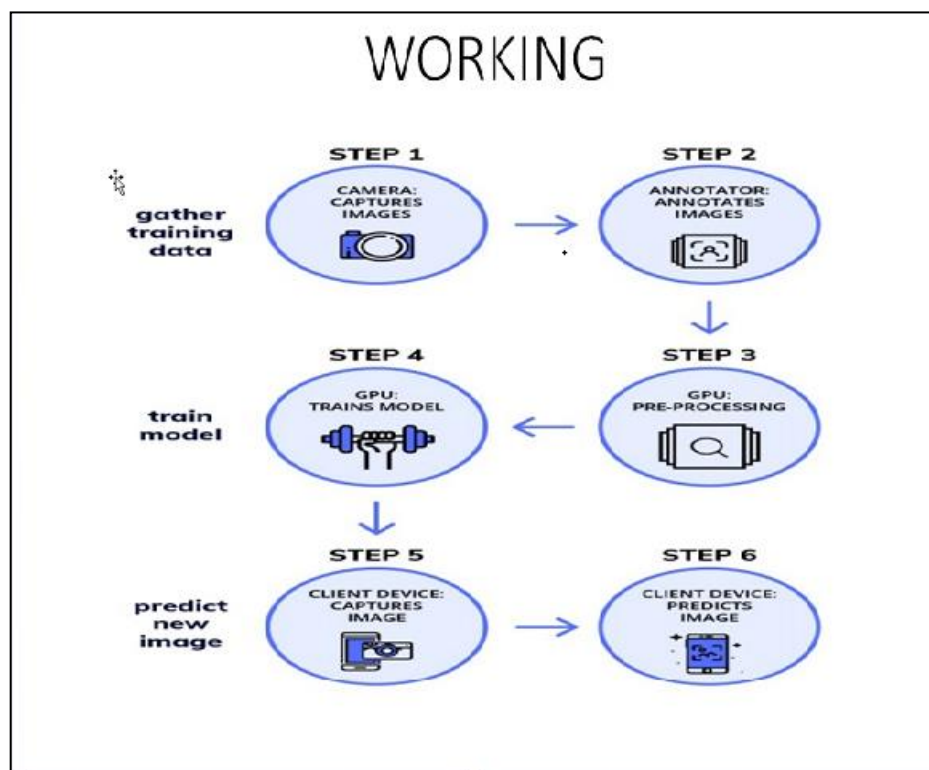


Figure 2.1 Block Diagram

Step1: The first step involves capturing the images for training, we have collected 150 images of cup, flask, jug as our dataset, which were later split into test images and train images

Step 2: After collecting the dataset we have to label the images further in order to teach the machine, this is performed by using a software tool called Lableimg.

Step 3: After labeling, particular algorithm should be chosen for the training purpose, we have chosen SSD MOBILE NET MODEL v2, which is made to run with 2000 steps in i5 processor machine.

Step 4: After successful training the test images were given so as to understand the model's accuracy and speed, in our case the chosen model has provided us with an accuracy of 96% and has detected all the three labelled datasets.

Step 5: After successful test results, now the real time object detection should be carried out. Through the webcam in our machine we were able to detect the images with greater accuracy.

Step 6: The final step is to convert the model to TFLITE model so that it works efficiently on our low processing device Raspberry Pi, the end results were with same accuracy as obtained in i5 machine, but the frames per second is slightly less i.e 0.9 frames per second.

CHAPTER-3

HARDWARE COMPONENTS

3.1GENERAL DESCRIPTION

Raspberry Pi3 (Model B+): Raspberry Pi is a low-cost, basic computer that was originally intended to help spur interest in computing among school-aged children. All the hardware components of our project are mostly connected through the Raspberry Pi. The Raspberry Pi is contained on a single circuit board and features ports for:

- HDMI
- USB 2.0
- Composite video
- Analog audio
- Power
- Internet
- SD Card

The computer runs entirely on open-source software and gives students the ability to mix and match software according to the work they wish to do.

3.1.1 Pi camera module

The Pi camera module v2 replaced the original camera module. The Raspberry pi camera will act as like the inbuilt camera as like in laptops. It helps in capturing pictures, recording videos as per our wish and save those images and videos in the specified path.

3.1.2 Hardware

Taking into account the relatively high-performance requirements of image processing in general and the equipment currently available to the faculty, as a relatively inexpensive and powerful embedded platform the Raspberry Pi was an obvious choice. The hardware specifications taken into consideration for this work can be seen in Table 1.

Specifications	Raspberry Pi 3 B+
CPU type/speed	ARM Cortex-A53 1.4GHz
RAM size	1GB SRAM
Integrated Wi-Fi	2.4GHz and 5GHz
Ethernet speed	300Mbps
PoE	Yes
Bluetooth	4.2

Table 3.1 Raspberry Pi hardware specifications

Another contributing factor to this choice was the availability of the Pi camera module, which can capture high-definition video as well as stills and requires the user to simply update Raspberry's firmware to the latest version. It can easily be used in OpenCV based applications. Although using the officially supported camera module, which can be accessed through the MMAL and V4L APIs and is supported by numerous third-party libraries, any USB web camera can be used.



Figure 3.1 Raspberry Pi 3 B+ and Pi Camera Module

Raspberry Pi 3 Model B is single board computer. Its CPU speed ranges between 700MHZ and 1.2GHZ. It also has on board memory between 256MB and 1GB Ram. This is used at transmitter or user end. It is the heart of the system. OS installed on it is Raspbian. The Raspberry Pi 3 Model B is the latest version of the Raspberry Pi computer. The Pi isn't like your typical machine, in its cheapest form it doesn't have a case, and is simply a credit-card sized electronic board of the type you might find inside a PC or laptop but much smaller. The Raspberry Pi 3 Model B+ is the latest product in the Raspberry Pi 3 range, boasting an updated 64-bit quad core processor running at 1.4GHz with built-in metal heatsink, dual-band 2.4GHz and 5GHz wireless LAN, faster (300mbps) Ethernet, and PoE capability via a separate PoE HAT. The Raspberry Pi 3 Model B+ maintains the same mechanical footprint as both the Raspberry Pi 2 Model B and the Raspberry Pi 3 Model B+. It can still use all the favorite Raspbian or PIXEL software with this update. We should make sure to upgrade our Raspbian operating system installs to the latest version so that the firmware can support the new chips. Old SD cards from previous releases will not work without an upgrade.

3.1.3 ADDITIONAL HARDWARE SPECIFICATIONS:

- Chipset: Broadcom BCM2837
- CPU: 1.2GHz quad-core 64-bit ARM cortex A53
- Ethernet: 10/100 (Max throughput 100Mbps)
- USB: Four USB 2.0 with 480Mbps data transfer
- Storage: MicroSD card or via USB-attached storage
- Wireless: 802.11n Wireless LAN (Peak transmit/receive throughput of 150Mbps), Bluetooth4.1
- Graphics: 400MHz Video Core IV multimedia
- Memory: 1GB LPDDR2-900 SDRAM
- Expandability: 40 general purpose input-output pins
- Video: Full HDMI port
- Audio: Combined 3.5mm audio out jack and composite video
- Camera interface (CSI)
- Display interface (DSI)

If Raspbian OS is not installed using the Noobs installer, and they may result in “running out of space”, they can also go into the terminal and type 'sudo raspi-config' and then select the option to 'Expand root partition to fill SD card', which will ensure that current available space on the card can also be used.

3.2 Pin Diagram of Raspberry Pi 3B+

RASPBERRY PI 3B+ is a development board in PI series. It can be considered as a single board computer that works on LINUX operating system. The board not only has tons of features it also has terrific processing speed making it suitable for advanced applications. PI board is specifically designed for hobbyist and engineers who are interested in LINUX systems and IOT (Internet of Things).

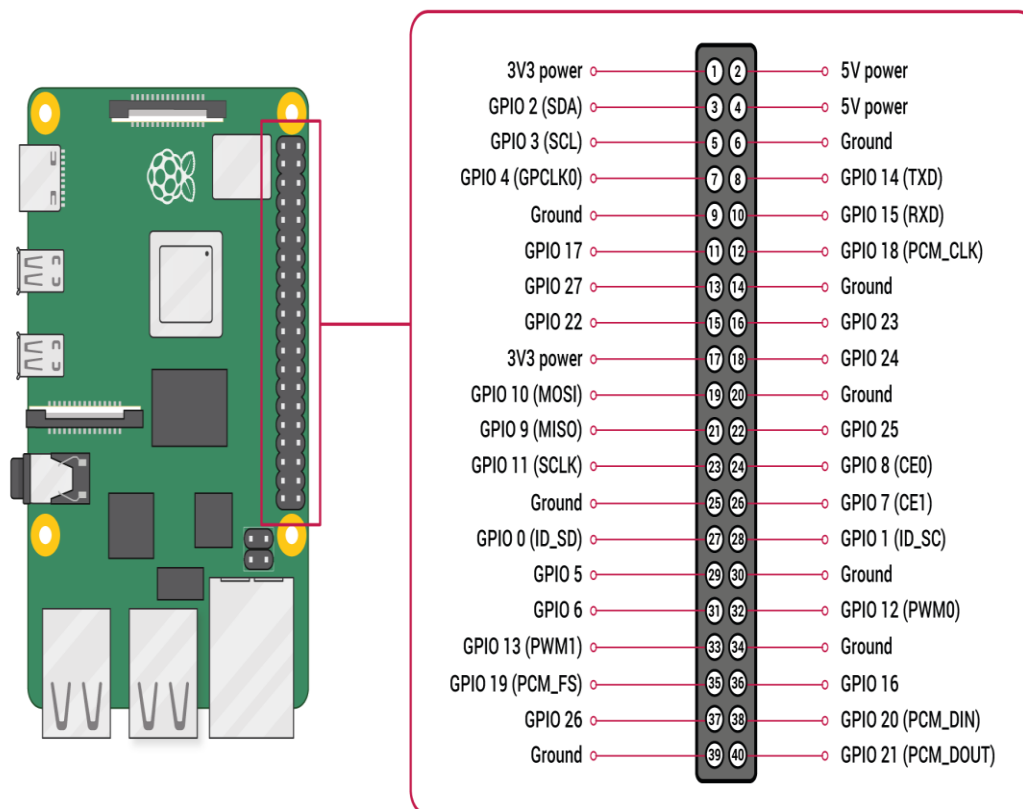


Figure 3.2 Pin diagram of Raspberry Pi 3 B+ Model

There are 40 Pins in Raspberry Pi 3 B+, where there are 7 Pin groups namely

- Power source
- Communication Interface
- SPI Interface
- TWI Interface
- Input output Pins
- PWM
- External Interrupts

Table 3.2 PIN SPECIFICATIONS OF RASPBERRY Pi 3B+ Model

PIN GROUP	PIN NAME	DESCRIPTION
POWER SOURCE	+5V, +3.3V, GND and Vin	+5V -power output +3.3V -power output GND – GROUND pin
COMMUNICATION INTERFACE	UART Interface (RXD, TXD) [(GPIO15, GPIO14)]	UART (Universal Asynchronous Receiver Transmitter) used for interfacing sensors and other devices.
SPI Interface (MOSI, MISO, CLK, CE) x2 [SPI0-(GPIO10,GPIO9, GPIO11 ,GPIO8)] [SPI1--(GPIO20, GPIO19, GPIO20, GPIO7)]	SPI (Serial Peripheral Interface) used for communicating with other boards or peripherals.	
TWI Interface (SDA, SCL) x2 [(GPIO2,GPIO3] [(ID_SD, ID_SC)]	TWI (Two Wire Interface) Interface can be used to connect peripherals.	
INPUT OUTPUT PINS	26 I/O	Although these some pins have multiple functions they can be considered as I/O pins.
PWM	Hardware PWM available on GPIO12, GPIO13, GPIO18, GPIO19	These 4 channels can provide PWM (Pulse Width Modulation) outputs. Software PWM available on all pins
EXTERNAL INTERRUPTS	All I/O	In the board all I/O pins can be used as Interrupts.

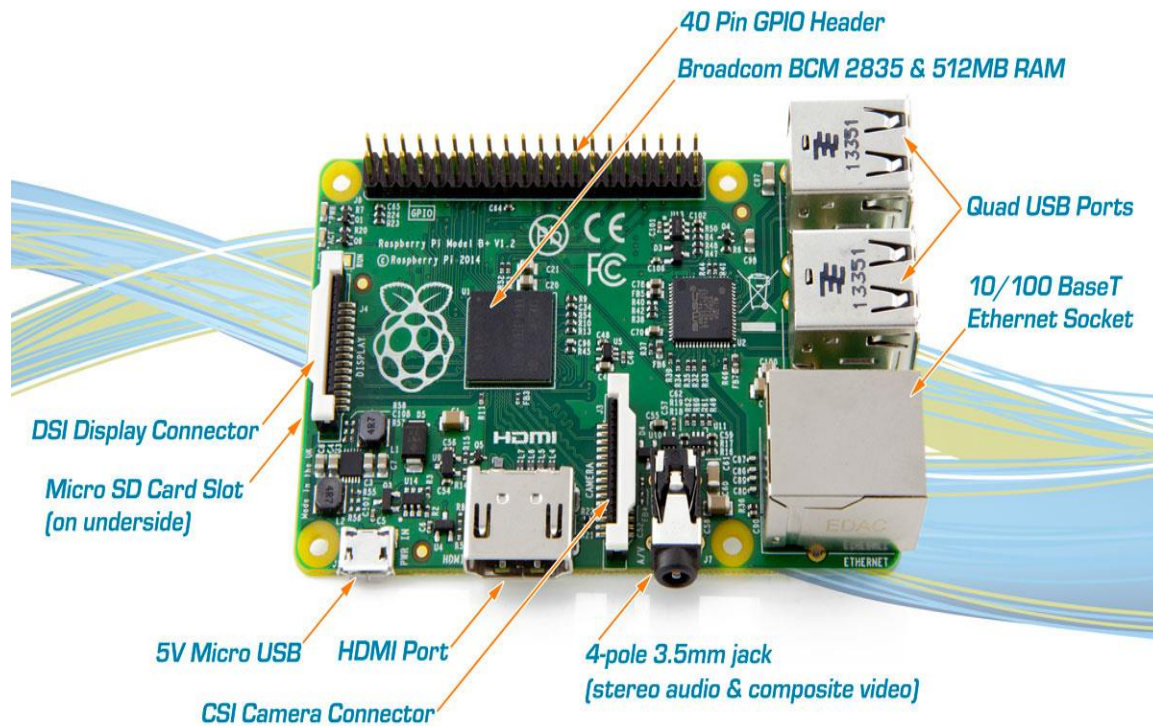


Figure 3.3 Blocks of Raspberry Pi 3 model

3.3 Webcam

A **webcam** is a video camera that feeds or streams an image or video in real time to or through a computer to a computer network, such as the Internet. Webcams are typically small cameras that sit on a desk, attach to a user's monitor, or are built into the hardware. Webcams can be used during a video chat session involving two or more people, with conversations that include live audio and video. For example, Apple's iSight camera, which is built into Apple laptops, iMacs and a number of iPhones, can be used for video chat sessions, using the iChat instant messaging program (now called Messages). Webcam software enables users to record a video or stream the video on the Internet.

As video streaming over the Internet requires a lot of bandwidth, such streams usually use compressed formats. The maximum resolution of a webcam is also lower than most handheld video cameras, as higher resolutions would be reduced during transmission.

The lower resolution enables webcams to be relatively inexpensive compared to most video cameras, but the effect is adequate for video chat sessions. The term "webcam" (a clipped compound) may also be used in its original sense of a video camera connected to the Web continuously for an indefinite time, rather than for a particular session, generally supplying a view for anyone who visits its web page over the Internet. Some of them, for example, those used as online traffic cameras, are expensive, rugged professional video cameras.

3.3.1 Interface

Typical interfaces used by articles marketed as a "webcam" are USB, Ethernet and IEEE 802.11 (denominated as IP camera). Further interfaces such as e.g. Composite video or S-Video are also available. The USB video device class (UVC) specification allows inter-connectivity of webcams to computers without the need for proprietary device drivers.

Characteristics

- Low manufacturing cost
- High flexibility, making them the lowest-cost form of videotelephony. As webcams evolved simultaneously with display technologies, USB interface speeds and broadband internet speeds, the resolution went up from gradually 320×240, to 640×480, and some even offering 1280×720 (aka 720p) or 1920×1080 (aka 1080p) resolution.
- Low-end webcams offering resolutions of 720p,
- Mid-range webcams offering 1080p resolution, and
- High-end webcams offering 4K resolution at 60 fps.

3.3.2 Uses

The most popular use of webcams is the establishment of video links, permitting computers to act as videophones or videoconference stations. Other popular uses include security surveillance, computer vision, video broadcasting, and for recording social videos. The video streams provided by webcams can be used for a number of purposes, each using appropriate software.

3.4 RASPBERRY PI CAMERA

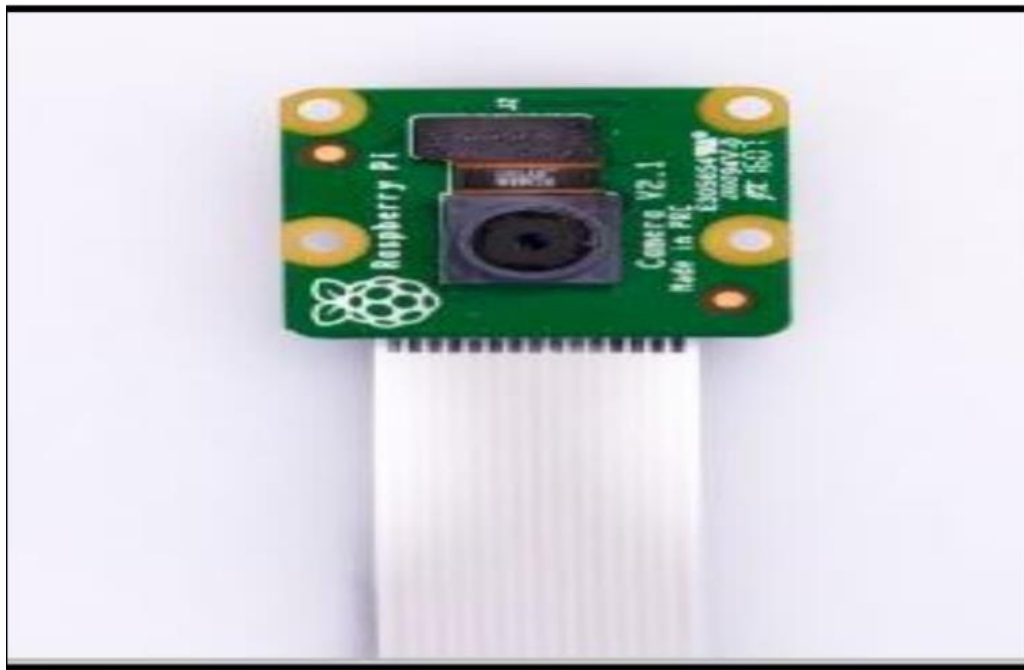


Figure 3.4 Raspberry pi camera

The Pi camera comes with a flex cable. The flex cable is inserted into the connector which is located between the Ethernet and HDMI port with the silver connectors facing the HDMI port. The flex cable connector is opened by pulling the tabs on the top of the connector upwards then towards the Ethernet port. The flex cable then is inserted firmly into the connector. The top part of the connector then is pushed towards the HDMI connector and down, while the flex cable is held in place.

Here the Pi camera is being utilized for the face detection and the face recognition process where firstly are captured and stored it in the database using python and then again using the camera while the automation and the surveillance part. The Raspberry Pi Camera Module v2 is a high quality 8 megapixel Sony IMX219 image sensor custom designed add-on board for Raspberry Pi, featuring a fixed focus lens. It's capable of 3280 x 2646 images static images, and also supports 1080p30, 720p60 and 640x480p60/90 video. It attaches to Pi by way of one of the small sockets on the board upper surface and uses the dedicated CSI interface, designed especially for interfacing to cameras.

3.4.1 SPECIFICATIONS

- 8 mega pixel camera capable of taking photographs of 3280 x 2464 pixels
- Capture video at 1080p30, 720p60 and 640x480p90 resolutions
- All software is supported within the latest version of Raspbian Operating System

3.4.2 APPLICATIONS

- CTV security camera
- Motion detection
- Time lapse photography

CHAPTER-4

SOFTWARE REQUIREMENT

4.1 GENERAL

The Raspberry Pi Foundation provides Raspbian, a Debian-based (32-bit) Linux distribution for download, as well as third-party Ubuntu, Windows 10 IoT Core, RISC OS. It promotes Python and Scratch as the main programming languages, with support for many other languages. The default firmware is closed source, while an unofficial open source is available. Many other operating systems can also run on the Raspberry Pi. Third-party operating systems available via the official website include Ubuntu MATE, Windows 10 IoT Core, RISC OS and specialized distributions for the Kodi media center and classroom management. The formally verified microkernel seL4 is also supported.

4.1.1 VNC Viewer

A virtual network computing system is platform dependent. This means that the client working on one type of operating system can't connect to the VNC server that operates on a different type of operating system. There are many different types of clients and servers available for different GUI based interfaces. In addition, a virtual network computing system is available for Java. Some of the VNC programs only work for the Windows operating system. VNC was originally developed by an AT&T Research Team, but virtual network computing systems are extremely popular in handling business and personal uses.

VNC works by transmitting all of your keyboard and mouse movements from your *thin* client computer to the other, *large* client computer. VNC can allow you to use an older computer to run a recent program. This sometimes allows businesses to purchase fewer computers. Businesses no longer need the same type of disk memory or processing capability, for VNC makes things easier. In addition to saving data, the benefits include saving time and money.

4.1.1.1 Uses of VNC

People can use VNC to remotely access files on certain computers in a wide range of situations.

- VNC can be effectively used by people who need to access work files from their office computer. It is also great for people who happened to forget to bring their presentation to work. They can use VNC to connect to their home computer in order to retrieve any files.
- This type of program is often used by system administrators, IT support, and help desks. It is used by administrators in order to take control of an employee's computer. Administrators may need to do this in order to help certain employees with any type of program.
- It can also be used in the classroom by teachers. Teachers may take advantage of this program in order to allow students to view what is happening at the teacher's computer. This can help teachers teach their students easily and more effectively.

4.1.2 PuTTY

PuTTY is an open-source application making use of network protocols like including SCP, SSH Telnet and rlogin in Windows and UNIX platforms in conjunction with an X term terminal emulator.

Over a network, PuTTY makes use of all the above protocols to enable a remote session on a computer. It is a popular tool for text-based communication and is also a popular utility for connecting Linux servers from Microsoft operating system-based computers.

4.1.2.1 Purpose of using PuTTY

Putty (software) generally has two purposes

4.1.2.1.1 Used as a File Transfer Protocol.

Most of the hosting services, both online and offline are built on LINUX OS, rightly so because it provides better safety for client data. Especially when thousands of client's data stored in a single place safety is the first priority. However, this poses a greater challenge for non-LINUX OS users to deal with. Here comes the third-party applications like PuTTY which enables non-Linux users install this particular software (PuTTY), and interact with Linux servers from a non-Linux OS. Interface of PuTTY is similar to windows terminal; however, user need to be aware of Linux commands to interact with its PuTTY provides various File transfer features like FTP and SFTP depending on user's security requirements.

4.1.2.1.2 Used to generate Hash key

PuTTY also used to generate SSH key. Nowadays using passwords are prone to security threats especially when you are dealing with lot of confidential data online. PuTTY allows you to generate a series of keys, which is a combination of hundreds of Alpha Numeric and special characters, which is almost impossible to crack. SSH generates two types of key combination. Public key which is used to access the terminal by authorised people and Private key which should not be shared with anyone. Private key is encrypted into the particular server of the user which can only be opened with a public key.

When you are using a particular system to access a server you can just point to the public key and you can login to the server any time without entering any password or user name. One place where particularly found the use of PuTTY is “Digital Ocean hosting services”.

4.1.3 OPENCV

OpenCV was started at Intel in 1999 by Gary Bradsky, and the first release came out in 2000. Vadim Pisarevsky joined Gary Bradsky to manage Intel's Russian software OpenCV team. In 2005, OpenCV was used on Stanley, the vehicle that won the 2005 DARPA Grand Challenge. Later, its active development continued under the support of Willow Garage with Gary Bradsky and Vadim Pisarevsky leading the project. OpenCV now supports a multitude of algorithms related to Computer Vision and Machine Learning and is expanding day by day. OpenCV supports a wide variety of programming languages such as C++, Python, Java, etc., and is available on different platforms including Windows, Linux, OS X, Android, and iOS. Interfaces for high-speed GPU operations based on CUDA and OpenCL are also under active development. OpenCV-Python is the Python API for OpenCV, combining the best qualities of the OpenCV C++ API and the Python language.

4.1.4 OPENCV PYTHON

OpenCV-Python is a library of Python bindings designed to solve computer vision problems. Python is a general-purpose programming language started by Guido van Rossum that became very popular very quickly, mainly because of its simplicity and code readability. It enables the programmer to express ideas in fewer lines of code without reducing readability. Compared to languages like C/C++, Python is slower. That said, Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules.

This gives us two advantages: first, the code is as fast as the original C/C++ code (since it is the actual C++ code working in background) and second, it is easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation. OpenCV-Python makes use of NumPy, which is a highly optimized library for numerical operations with a MATLAB-style syntax. All the OpenCV array structures are converted to and from NumPy arrays. This also makes it easier to integrate with other libraries that use NumPy such as SciPy and Matplotlib.

4.1.5 TENSORFLOW LITE

When a ML model is implemented and trained using TensorFlow, you usually end up with a model file that requires ample storage space and a GPU to run inference. Luxuries such as large storage space and GPUs are not available on most mobile devices. TensorFlow lite is the solution to enabling ML models within mobile devices.

TensorFlow Lite models exist within the devices and have low latency, which is essential for real-time inference application, as network round trips from mobile devices to cloud servers where models are hosted might test the patience of your application users. TensorFlow Lite takes existing TensorFlow models and converts them into an optimized and efficient version in the form of a .tflite file. The streamlined model is small enough to be stored on devices and sufficiently accurate to conduct suitable inference.

TensorFlow Lite is used in:

- Mobile devices are prime devices to utilize the TensorFlow Lite model. TensorFlow Lite provides a variety of pre-trained models that can be easily converted to .tflite versions and integrated with mobile applications in their dedicated development environments.

- Internet of Things Devices
- Raspberry pi

4.1.6 RASPBIAN OS

Raspbian is a Debian operating system of Raspberry pi. There are several versions of Raspbian including Raspbian Buster and Raspbian Stretch. Since 2015 it has been officially provided by Raspberry pi foundation, as the primary operating system for the family of Raspberry Pi as a single board computer. Raspbian was created by Mike Thompson and Peter Green as an independent project. The initial build was completed in June 2012. The operating system is still under active development. Raspbian is highly optimized for the Raspberry Pi line's low-performance ARM CPUs. Raspbian uses PIXEL, **Pi Improved X-Window Environment, Lightweight** as its main desktop environment as of the latest update. It is composed of a modified LXDE desktop environment and the Open Box stacking window manager with a new theme and few other changes. The distribution is shipped with a copy of the algebra program Wolfram and Mathematica and a version of Minecraft called Minecraft Pias well as a lightweight version of Chromium as of the latest version. It is composed of a modified LXDE desktop environment and the Open Box stacking window manager with a new theme and few other changes. The initial build was completed in June 2012. The operating system is still under active development.

2017-08-17	2017-08-16	9 (Stretch)			1.4.6		✓	✓	✓	✓	✗	✗
2017-09-08	2017-09-07						✓	✓	✓	✓	✗	✗
2017-11-29	2017-11-29						✓	✓	✓	✓	✗	✗
2018-03-13	2018-03-13						✓	✓	✓	✓	✗	✗
2018-04-18	2018-04-18		4.14	6.3	1.4.8		✓	✓	✓	✓	✓	✗
2018-06-29	2018-06-27						✓	✓	✓	✓	✓	✗
2018-10-09	2018-10-09						✓	✓	✓	✓	✓	✗
2018-11-13	2018-11-13						✓	Yes	✓	✓	✓	✗
2019-04-08	2019-04-08						1.4.9	✓	✓	✓	✓	✓
2019-06-24	2019-06-20	10 (Buster)	4.19	8.3	1.8.2	✓	✓	✓	✓	✓	✓	
2019-07-10	2019-07-10					✓	✓	✓	✓	✓	✓	
2019-09-30	2019-09-26					✓	✓	✓	✓	✓	✓	
2020-02-07	2020-02-05					✓	✓	✓	✓	✓	✓	
2020-02-14	2020-02-13					✓	✓	✓	✓	✓	✓	
Release Date	Release Name	Debian Version	Linux Kernel	GCC	apt	X Server	Pi 1/1+	Pi 2	Pi 3	Pi Zero W	Pi 3+	4

Figure 4.1 Stretch Version and its compatibility of different Raspberry Pi's

The stretch version has been chosen according to the specifications of the raspberry pi 3 B+.

4.2 INSTALLATION OF RASPBIAN OPERATING SYSTEM ON RASPBERRY Pi

Installing Raspbian on the Raspberry Pi is really clear. Raspbian will be downloaded and writing the disc image to a micro SD card, at that point booting the Raspberry Pi to that microSD card. For this undertaking, one needs a microSD card (with no less than 8 GB), a PC with a space for it, and, obviously, a Raspberry Pi and fundamental peripherals (a mouse, console, screen, and power source). This isn't the main strategy for introducing Raspbian (more on that in a minute), yet it's a valuable method to learn on the grounds that it can likewise be utilized to introduce such a significant number of other working projects on the Raspberry Pi. When one realizes how to compose a circle picture to a microSD card, we open up a great deal of alternatives for Raspberry Piprojects.

Step1: Download the Raspbian

Turn on the PC and download the Raspbian disc image. One can locate the most recent variant of Raspbian on the Raspberry Pi Foundation's site here. It will take some time, particularly in the event when one intends to utilize the conventional download alternative as opposed to the other download sources. It can take a stretch of half hour or more to download. Choose Raspbian Stretch with Desktop if you want to have access to the Raspbian GUI; in other words, if you want to log in and be able to access a desktop, icons, etc. like you would with Windows or MacOS. Choose Raspbian Stretch Lite if you only need to boot to the command line. For simpler Raspberry Pi projects, this is often a good choice since the Lite version uses less power and fewer resources.

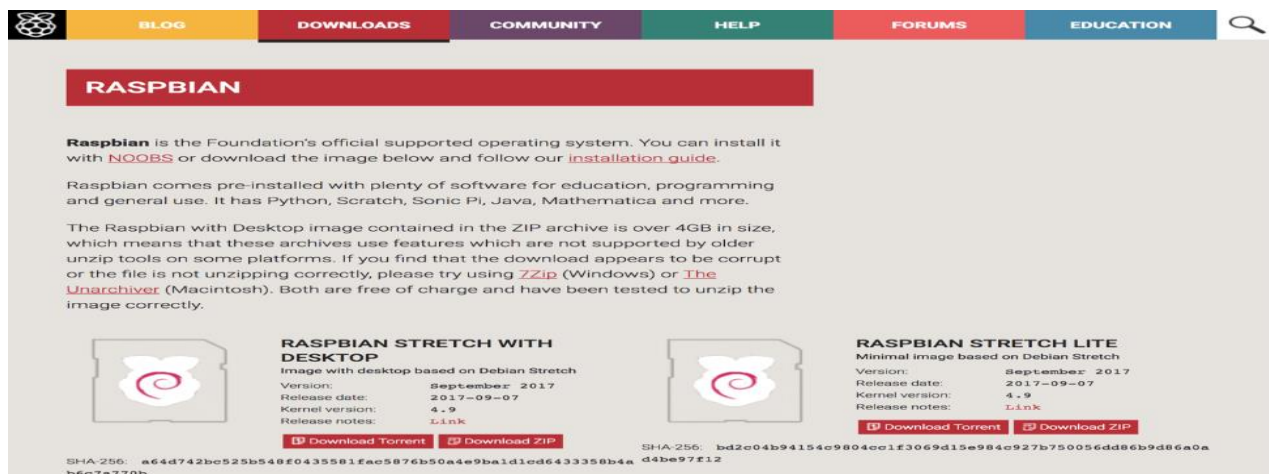


Figure 4.2 Downloading Raspbian Stretch

Step2: Unzip the file

The Raspbian disc image is compressed, so it should be unzipped. The file uses the ZIP64 format, so depending on how current built-in utilities are, one needs to use certain programs to unzip it. Linux users will use the appropriately named Unzip.

Step 3: Use Etcher

One has to pop the microSD card into our computer and write the disc image to it. The process of actually writing the image will be slightly different across these programs, but it's pretty self-explanatory no matter what is being used. Each of these programs will have us select the destination and the disc image (the unzipped Raspbian file). Choose, double-check, and then button to write.

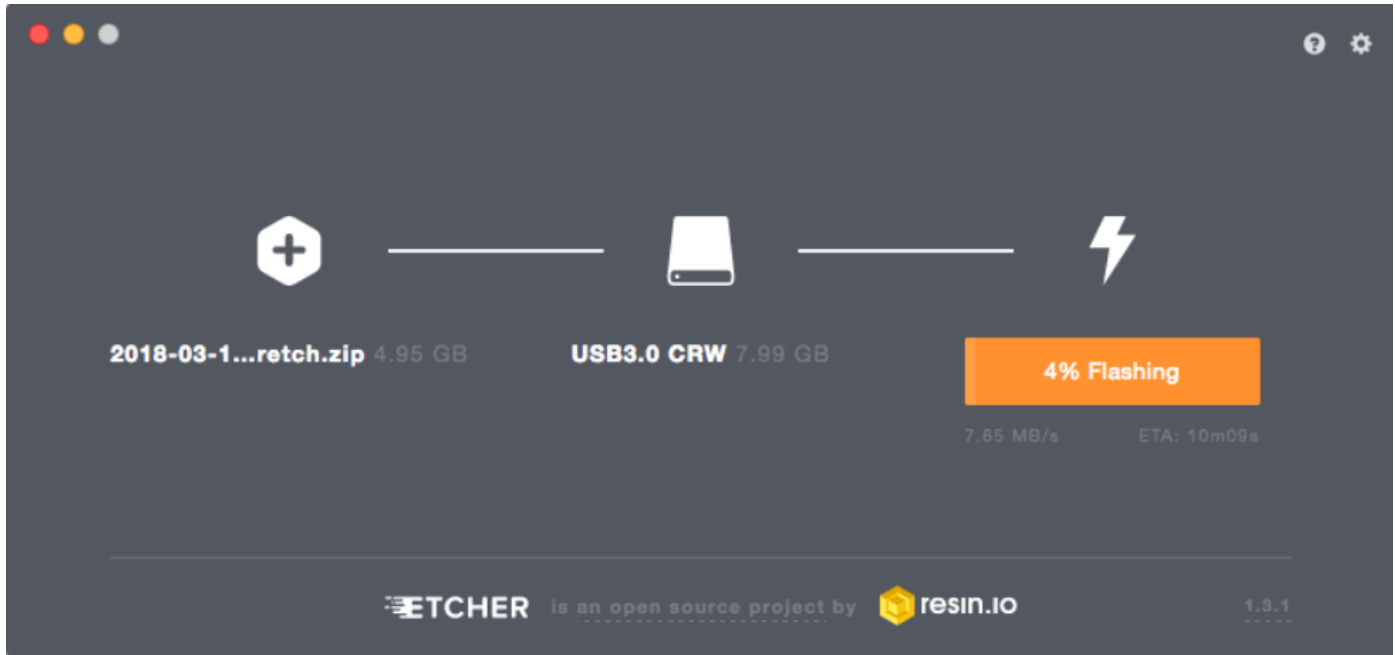


Figure 4.3 Using Etcher for flashing

The easiest way to flash Raspbian Stretch to your SD card is to download and install Etcher. After opening Etcher, select the Raspbian disk image, your SD card, and click Flash. After Etcher finishes running, the process ends.

Step 4: Put the microSD card in your Pi and boot up

When the disc image has been kept in touch with the microSD card, it is prepared to go. Put that microSD into your Raspberry Pi, plug in the peripherals and power source. The present release to Raspbian will boot straightforwardly to the desktop. Our default credentials are username pi and password raspberry.

Step 5:

Connect the Raspberry Pi to your network using a network cable, connect the keyboard and mouse and plug in the HDMI cable between the Pi and the Desktop.

Step 6:

After the Pi finishes booting up then the Raspbian desktop should appear. Open the Raspberry Menu, go to Preferences, then open Raspberry Pi Configuration

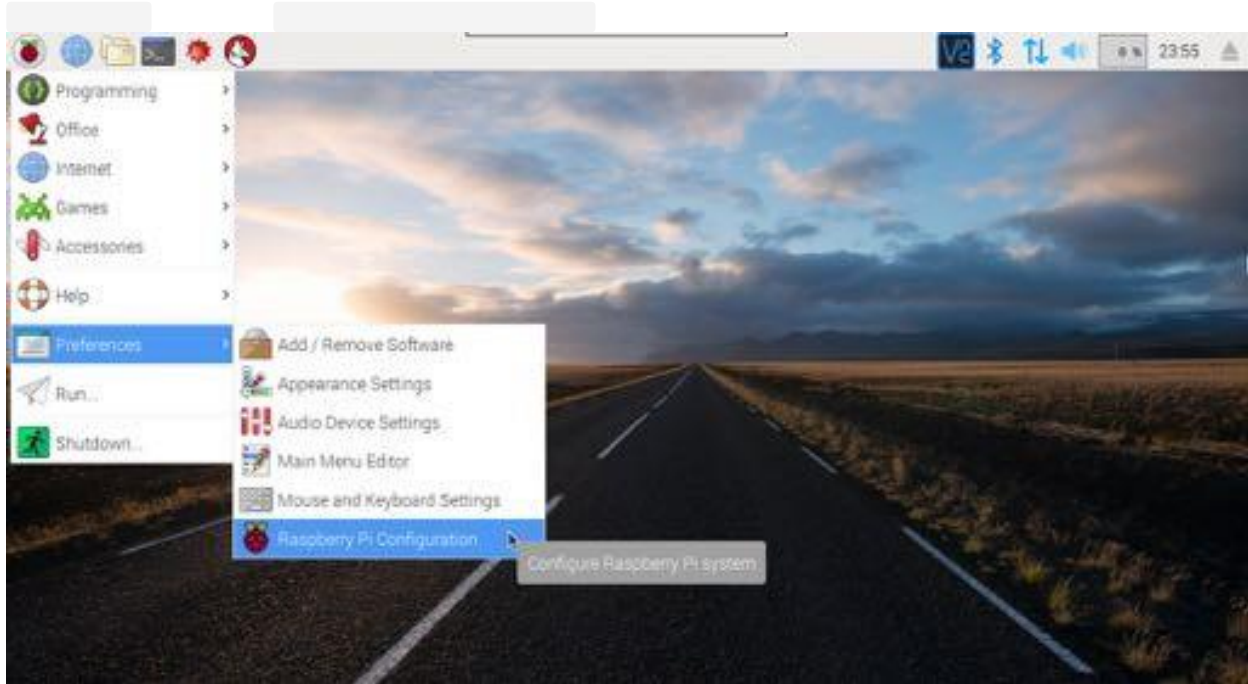


Figure 4.4 Opening Raspberry Pi Configuration menu

STEP 7:

Check Wait for Network. This means the Pi will wait to start the desktop, and XLink, until we have a network connected

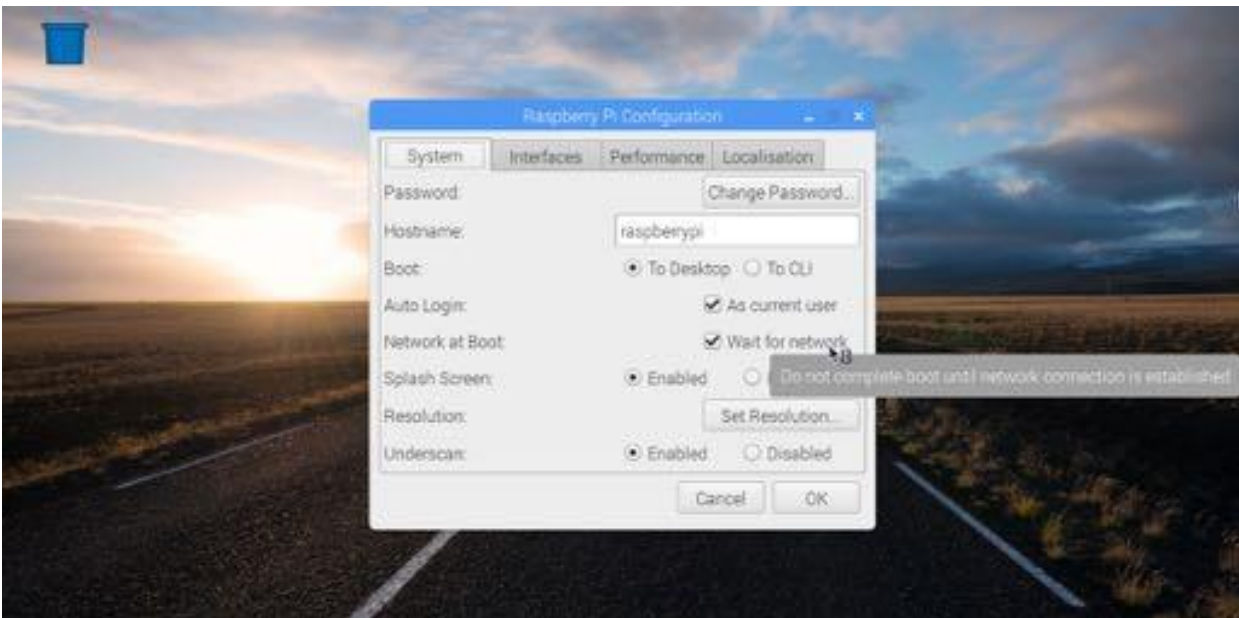


Figure 4.5 Enabling System in Raspberry Pi Configuration

STEP 8:

Enable SSH and VNC, this will enable us to access the Raspberry Pi remotely without needing a TV/Keyboard later on.

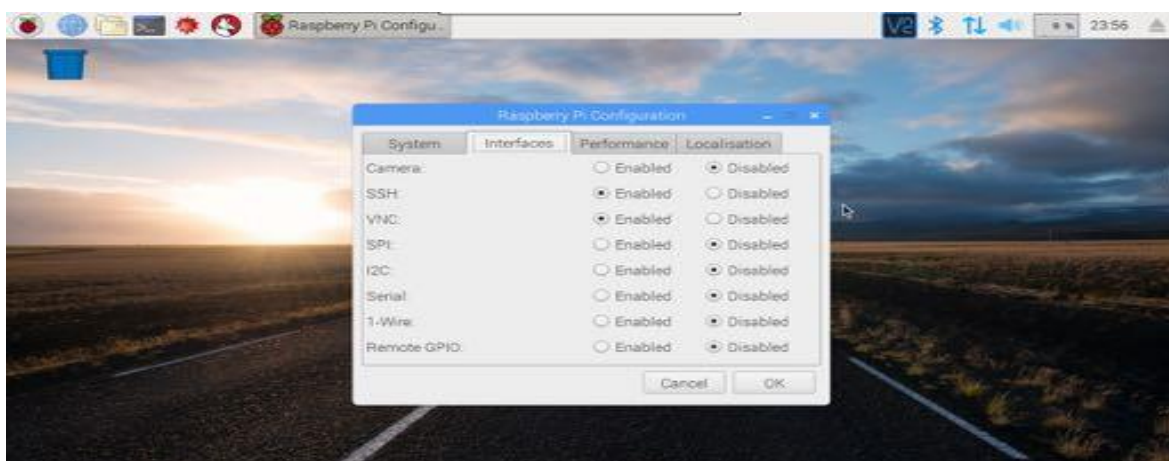


Figure 4.6 Enabling SSH and VNC in Raspberry Pi Configuration

STEP 9:

When prompted reboot the raspberry pi .

4.3 CAMERA MODULE CONFIGURATION IN RASPBERRY PI

Install the Raspberry Pi Camera module by inserting the cable into the Raspberry Pi camera port. The cable slots into the connector situated between the USB and micro-HDMI ports, with the silver connectors facing the micro-HDMI ports. Incase of confusion, just make sure the blue part of the cable is facing the USB ports on the Raspberry Pi:



Figure 4.7 Placing Cable in the Raspberry Pi camera port

Now boot the Raspberry Pi (plug the power in and turn it on).

Once booted, update the Raspberry Pi by running the following commands in a terminal window:

sudo apt-get-update

sudo apt-get upgrade

Now run the following command to go into the Raspberry Pi configuration tool:

sudo raspi-config

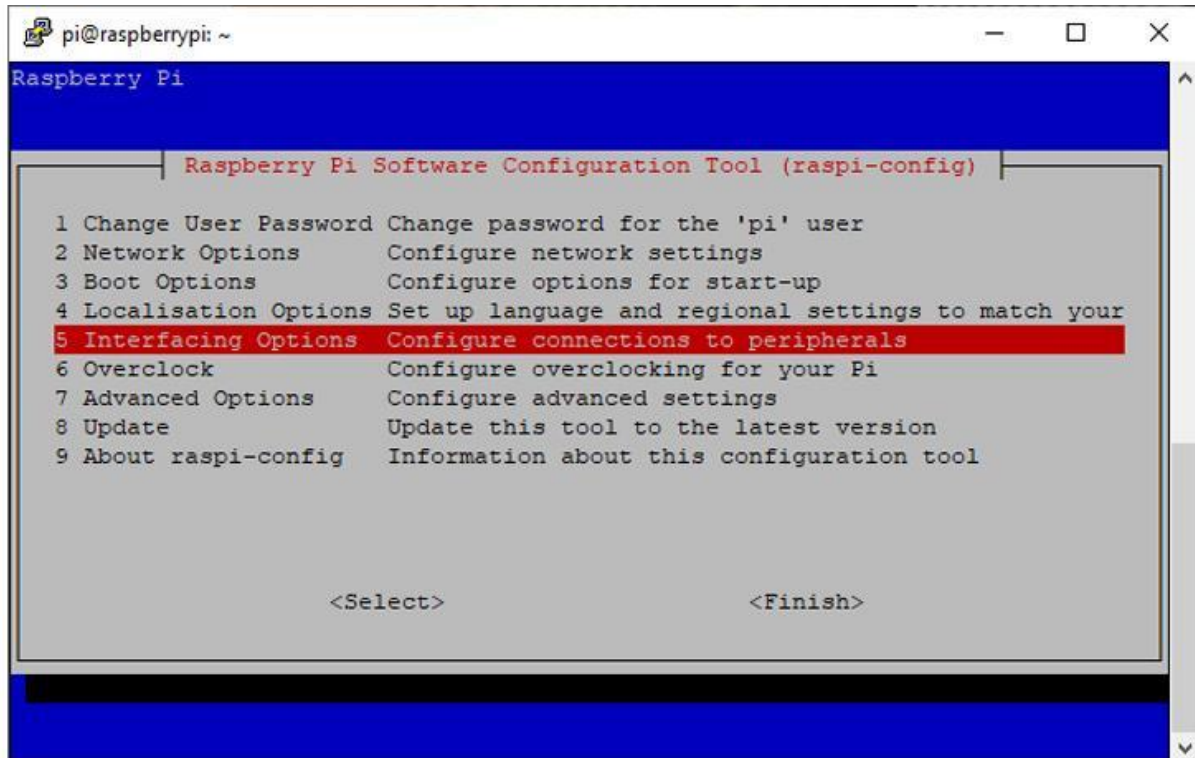


Figure 4.8 Selecting Interfacing Option in Configuration Tool

Navigate to Interfacing Options and hit Enter. Now select the 'Camera' option, and hit the Enter key to enable it. Select “Finish” and select to reboot the Raspberry Pi.

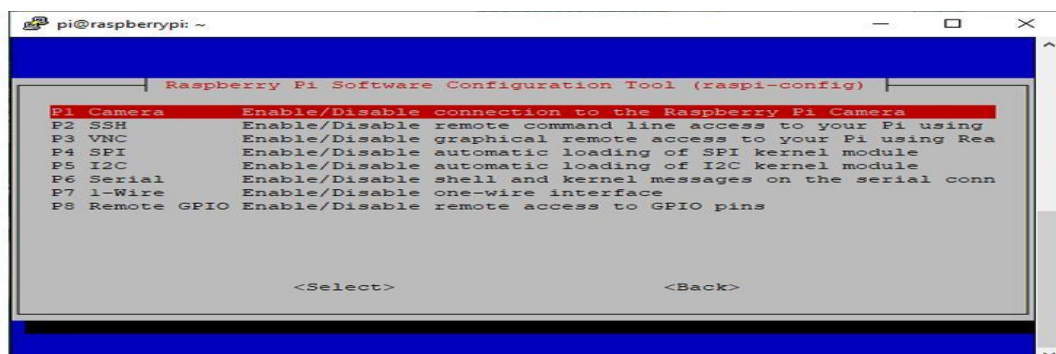


Figure 4.9 Enabling Pi Camera to establish connection

4.3.1 To Capture aimage with Raspberry Pi Camera Module

"Raspistill" is a command line application that allows to capture images with the camera module. To capture an image in jpeg format, type the following command into the terminal window:

```
raspistill -o image.jpg
```

Here, "image" is the name of the image that will be saved to the Raspberry Pi.

4.3.2 To record a video with Raspberry Pi Camera Module

"Raspivid" is a command line application that allows you to capture video with the camera module. To capture a 10 second video with the Raspberry Pi camera module, run the following command:

```
raspivid -o video.h264 -t 10000
```

Here, "video" is the name of the video and "10000" is the number of milliseconds

CHAPTER-5

TESTING AND TRAINING

5.1. TensorFlow

TensorFlow is an open source machine learning framework for all developers. It is used for machine learning and deep learning applications. To develop and research on fascinating ideas on artificial intelligence, Google team created TensorFlow. Google product uses machine learning in all of its products to improve the search engine, translation, image captioning or recommendations. It is designed to be executed on single or multiple CPUs and GPUs, making it a good option for complex deep learning tasks.

5.1.1. History of TensorFlow

As deep learning started to outperform all other machine learning algorithms when giving a massive amount of data. Google saw it could use these deep neural networks to improve its services:

- Gmail
- Photo
- Google search engine

They build a framework called TensorFlow to let researchers and developers work together on AI model. It was first made public in late 2015, while the first stable version appeared in 2017. It is open source under Apache Open Source license. We can use it, modify it and redistribute the modified version for free

5.1.2. TensorFlow Architecture:

TensorFlow architecture works in three parts

- Pre-processing the data
- Build the model
- Train and estimate the model

It is called TensorFlow because it takes input as a multi-dimensional array, also known as **tensors**. We can construct a sort of **flowchart** of operations called a Graph that we want to perform on that input. The input goes in at one end, and then it flow through this system of multiple operations and comes out the other end as output.

5.1.3. Running TensorFlow

We can train it on multiple machines like desktop, laptop etc., then we can run it on a different machine, after we trained the model. The model can be trained and used on GPUs as well as CPUs. Stanford researchers found that GPU was very good at matrix operations and algebra so that it makes them very fast for doing these kinds of calculations. Deep learning relies on a lot of matrix. TensorFlow is very fast at computing the matrix multiplication. A significant feature of TensorFlow is the Tensor Board, which enables to monitor graphically and visually what TensorFlow is doing.

5.1.4. Components of TensorFlow

1.Tensor:

TensorFlow's name is directly derived from its core framework: Tensor. In TensorFlow, all the computations involve tensors. A tensor is a vector or matrix of n-dimensions that represents all types of data. All values in a tensor hold identical data type with a known

shape. The shape of the data is the dimensionality of the matrix or array. A tensor can be originated from the input data or the result of a computation. In TensorFlow, all the operations are conducted inside a graph. The graph is a set of computation that takes place successively. Each operation is called an op code and are connected to each other. The graph outlines the ops and connections between the nodes. However, it does not display the values. The edges of the nodes is the tensor, is a way to populate the operation with data.

2. Graphs:

TensorFlow makes use of a graph framework. The graph gathers and describes all the series computations done during the training. The graph has lots of advantages:

- It was done to run on multiple CPUs or GPUs and even mobile operating system.
- The probability of the graph allows to preserve the computations for immediate or later use. The graph can be saved to be executed in the future.
- All the computations in the graph are done by connecting tensors together. A tensor has a node and an edge. The node carries the mathematical tensors together and produces an endpoints output. The edges explain the input/output relationships between nodes.

5.2. Training TensorFlow Object Detection Classifier

To detect the required object from each and every object given as input, can be done by training the object detection classifier with the set of input data. By this we will make the classifier to learn from the inputs and get trained for the detection in real time inputs. For this training of classifier, we will be using anaconda environment with python and TensorFlow.

5.2.1. Anaconda

Anaconda is a FOSS (Free and Open Source Software) distribution of the Python and R programming languages for scientific computing which includes machine learning. This software provides 7,500 + open source packages and we will be using some of these packages in this project

5.2.1.1. Download Anaconda

Anaconda is easy to use as we can create virtual environment for any python in this software. We will be downloading anaconda python package.

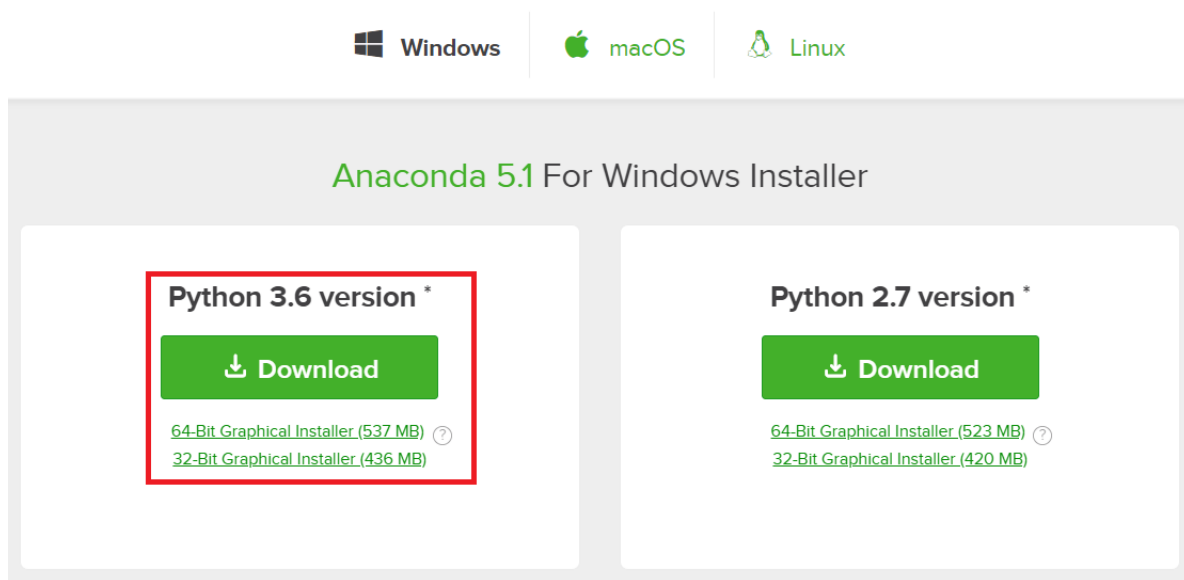


Figure 5.1 Download Python 3.6 version of Anaconda

5.2.1.2. Install Anaconda

Installing is very easy and quick once downloaded. Open the setup and follow the wizard instructions. Important thing is that it will automatically install python and some basic libraries with it.

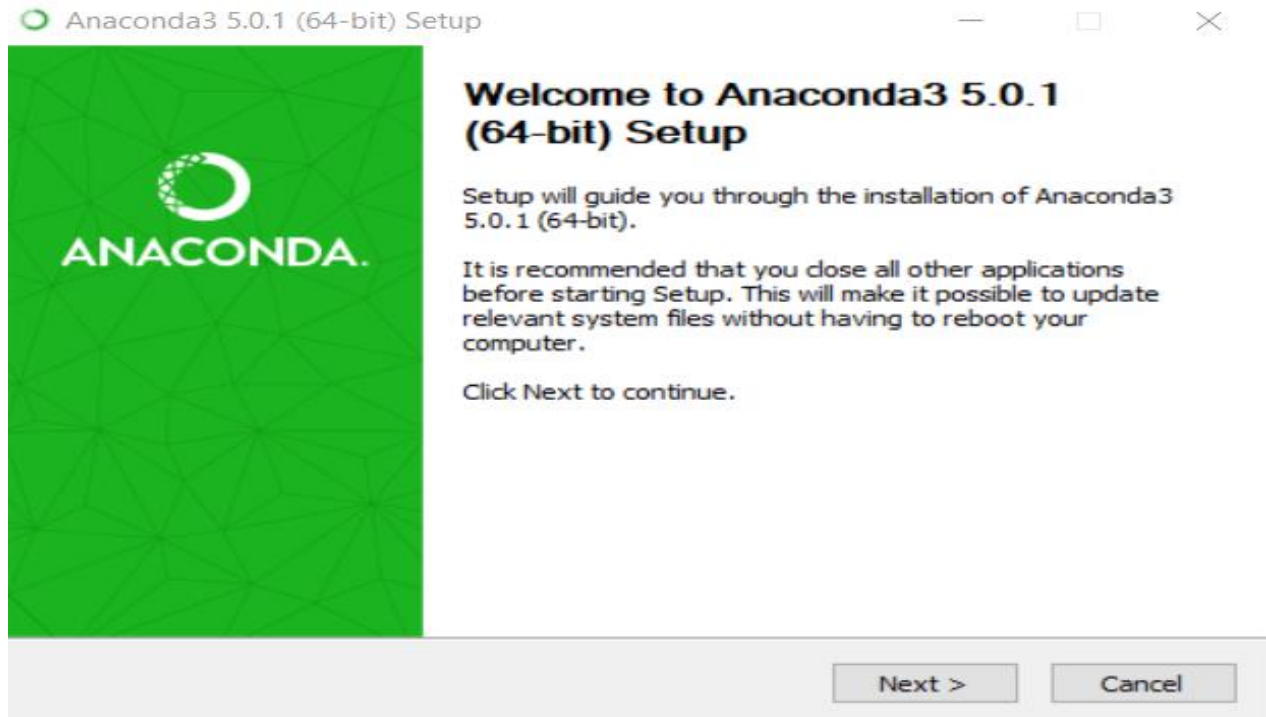


Figure 5.2 Installing Anaconda 3.5.0.1 version

5.2.2. Setting up the Object Detection directory Structure and Anaconda Virtual Environment:

5.2.2.1. TensorFlow Object Detection API

We created a folder named tensorflow1. This working directory will contain the full TensorFlow Object Detection framework as well as our training images, training

classifier, training data, trained classifier, configuration files and everything needed for object detection classifier.

```
C:\
└─
    Tensorflow1
        └─ models
            ├── official
            ├── research── object detection
            ├── samples
            └─ tutorials
```

The TensorFlow models repository's code which contains the object detection API is continuously updated by the developers. Sometimes they break functionality with old versions of TensorFlow. So it is best to use correct version of TensorFlow and download the suitable models repository. We in our project used TF v1.13 [1] version. This model is downloaded and placed in the tensorflow1 folder

5.2.2.2. SSD_MobileNet_v2_coco_model from TensorFlow's model zoo

TensorFlow provides several object detections models (pre trained classifiers with specific neural network architectures) in its model zoo [1]. This model zoo provides a collection of detection models pre-trained on the COCO dataset, the Kitti dataset and the other things. SSD-Mobile Net model have an architecture that allows for faster detection but with less accuracy but some models such as Faster-RCNN model give slower

detection but with more accuracy. We initially started with Faster-RCNN model and we are not able to convert the model in to tflite [1] for raspberry pi. So, we retrained our detector on the SSD-Mobile Net model and the detection worked considerably better. Later we are able to find that for using Object Detector on a device with low computational power (such as a smart phone or raspberry pi) use the SSD-Mobile Net model.

5.2.2.3. Anaconda virtual Environment:

We will create a new anaconda environment for our specific usage so that will not affect the base of anaconda. We created a virtual environment named tensorflow1 for our project and also installed python version 3.6.6 as this is the version which currently satisfies all our libraries which are used. Being Open Source there is many difficulties in setting the working lab environment

```
(tensorflow1) C:\User\ [redacted]
Python 3.6.6 [Anaconda, Inc.] (default, Jun 28 2018, 11:27:44) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Figure 5.3 Created virtual environment and python version

Using Pip Install Package command for TensorFlow we installed TensorFlow version 1.13. This is also we installed specifically as we use Object Detection API for our project. After clearing environment setting errors, we came to conclusion to use this version. After creating new virtual environment activate the environment update the pip

5.2.2.4. TensorFlow installation:

There exist two generic variants of TensorFlow, which utilize different hardware on our computer to run our computationally heavy Machine Learning algorithms. They are

TensorFlow CPU and TensorFlow GPU. As we do not have the GPU processors, we decided to make use of the CPU version of TensorFlow.

TensorFlow CPU which runs directly on the CPU of the machine and it is the slowest in terms of performance. Then install the other necessary packages in the table

Table 5.1 Versions specified Libraries

Name	Version
Pillow	7.0.0
Lxml	4.5.0
Cython	0.29.15
Contextlib2	0.6.0.post1
Jupyter	1.0.0
Matplotlib	3.2.1
Pandas	1.0.3
OpenCV-python	4.2.0.32

The pandas and OpenCV-python packages are not needed by TensorFlow, but they are used in the python scripts to generate TFRecords and to work with images, videos and webcam feeds

5.2.2.5. Configuring PYTHONPATH environment variable

A PYTHONPATH is an environment variable which can set to add additional directories where python will look for modules and packages. So, we created PYTHONPATH that points to the \models, \models\research, and \models\research\slim directories. We can add from any directory using this command

```
(tensorflow1) C:\> set
```

```
PYTHONPATH=C:
```

```
\tensorflow1\models;C:\tensorflow1\models\research;C:\tensorflow1\models\research\slim
```

5.2.3. Gathering and Labelling images

This step is very important in this project we have to give the images to learn. This is like the study material which we are providing for a student to learn and get good grades. Here also the same way we will collect all images and we will arrange and label them in a proper way and we will give that annotated images for the Neural Network to understand.

5.2.3.1. Gather images:

TensorFlow needs minimum hundreds of images for an object to train a good detection classifier. To train a robust classifier, the training images should have random objects in the image along with the desired objects, and should have a variety of backgrounds and different lighting conditions. There should be some images where the desired object is partially obscured, overlapped with something else or only halfway in the picture. For our project we have 3 different objects (the jug, flask, cup). We used our android phone to take about 40 images of each object and then we took about another 60 pictures with multiple objects in the picture. We know that to detect when objects are overlapping. So we took many images with overlapping

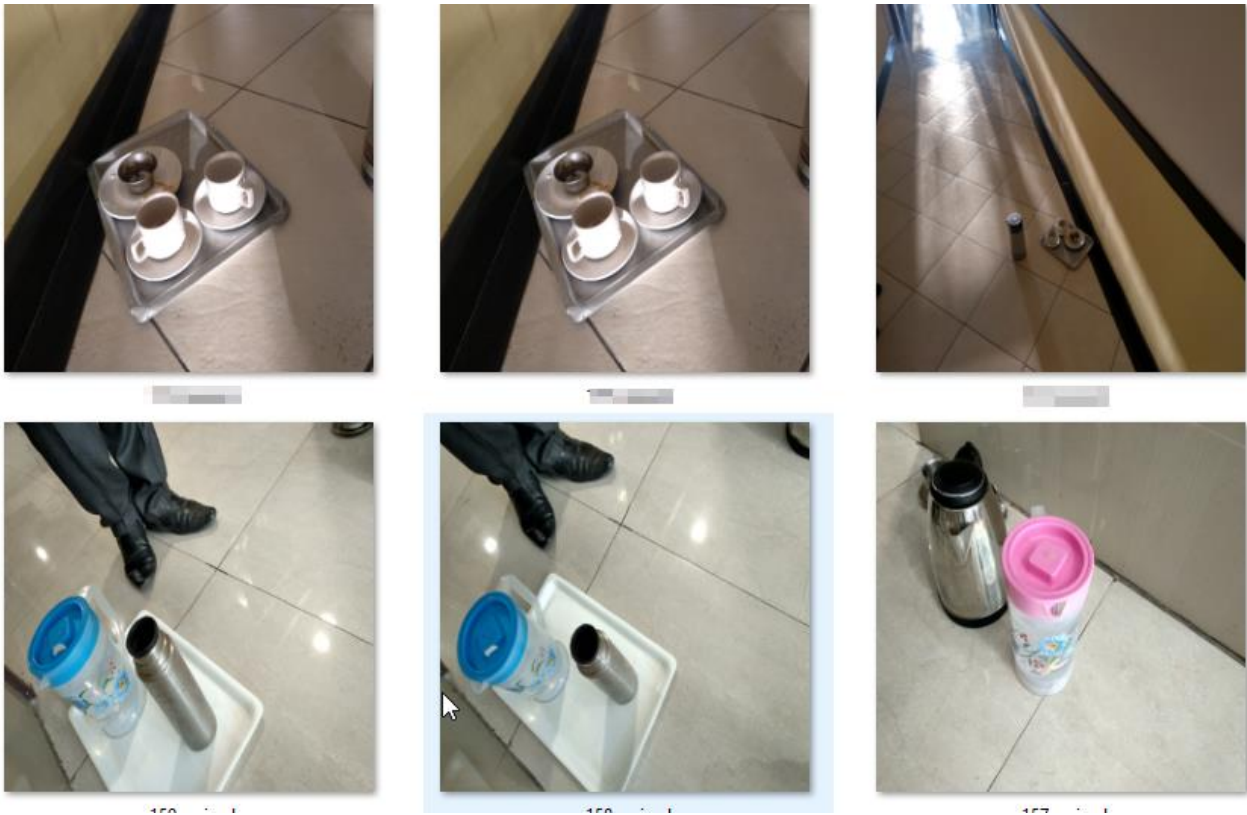


Figure 5.4 Gathering images for annotation

In this as all our images size was more than 3 megabyte per image it took around 800 seconds to complete 1 step in the training process ..so we decided to change all pictures size to 200 kilo bytes .To resize the images we used a script which is resizer.py .After we have all the pictures we need, move 20% of them to the \object_detection\images\test directory, and 80% of them to the \object_detection\images\train directory. Made sure there are a variety of pictures in both the \test and \train directories.

5.2.3.2. Label pictures:

With all the images gathered, it's time to label the desired objects in every picture. LabelImg [1] is a great tool for labeling images. This is a time-consuming process. LabelImg saves a .xml file containing the label data for each image. These .xml files will be used to generate TFRecords, which are one of the inputs to the TF trainer.

Once we have labeled and saved each image, there will be one .xml file for each image in the \test and \train directories.

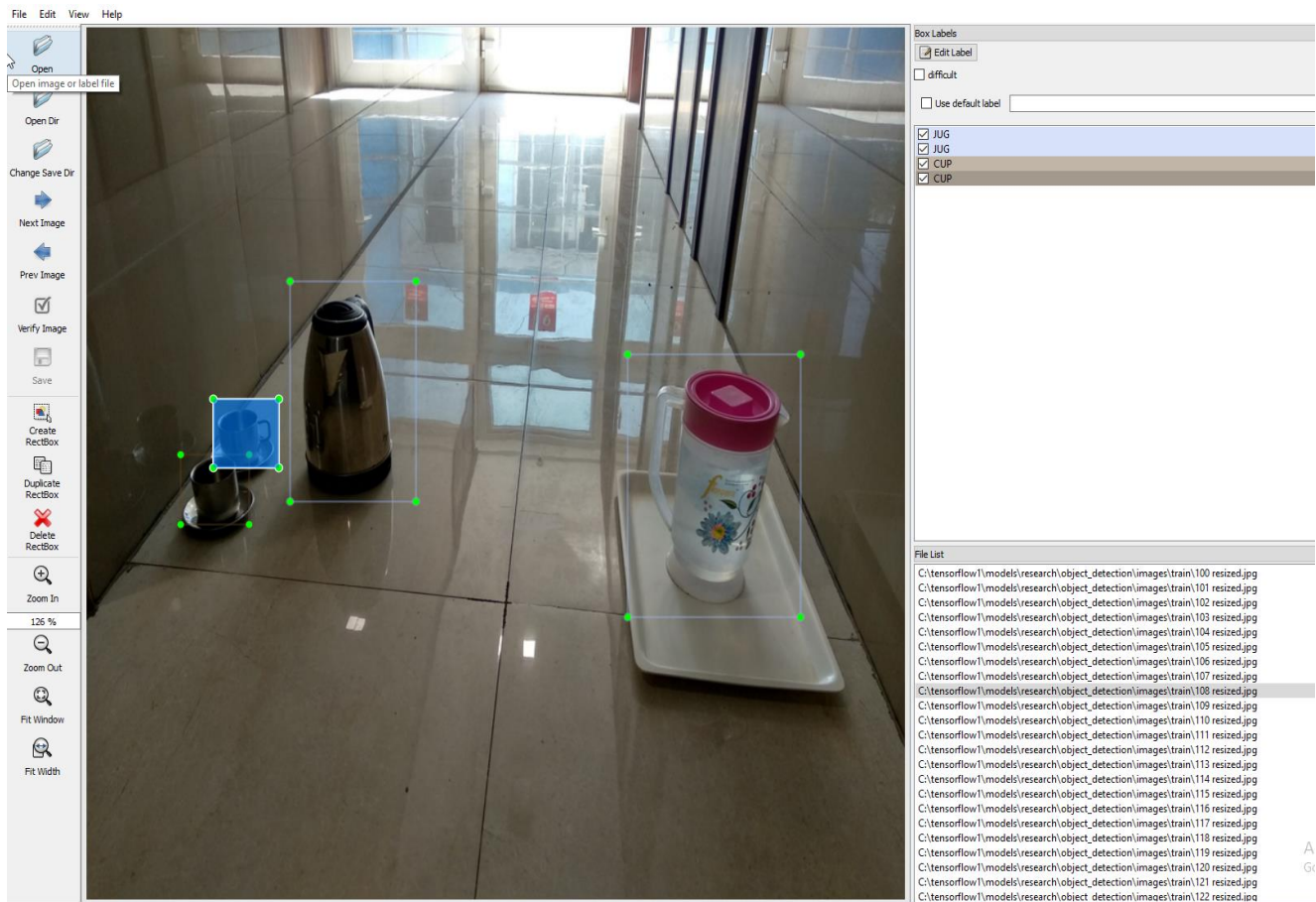


Figure 5.5 Labeling images using LabelImg

This is the LabelImg tool in this the open dir is used to open the train directory and the images in the directory are selected for the annotation. The suitable .xml file will be created for the .jpg file in the same directory. The xml file will contains the class name with the coordinates for that class name. Here class name is that the object name and the coordinates for that in the particular image is given in the xml file which would be used by the object detection classifier to learn from the group of .xml files which is the TFRecord.



Figure 5.6 XML content generated for animage

5.2.4. Generating Training Data

With the images labeled we generated the TFRecords that serve as input data to the TensorFlow training model. We used `xml_to_csv.py` and `generate_tfrecord.py` scripts to generate the training data

5.2.4.1. CSV file

First we will convert the .xml data to .csv files containing all the data for the train and test images.

```

(tensorflow1) C:\tensorflow1\models\research\object_detection>python xml_to_csv.py
Successfully converted xml to csv.
Successfully converted xml to csv.

```

Figure 5.7 XML is converted to CSV

This is the csv file from the above script.

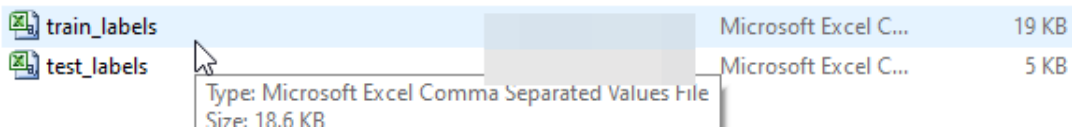


Figure 5.8 Generated csv file

1	filename	width	height	class	xmin	ymin	xmax	ymax	
2	100 resize	800	800	CUP	541	399	572	422	
3	100 resize	800	800	JUG	516	347	596	403	
4	101 resize	800	800	JUG	427	431	487	496	
5	101 resize	800	800	JUG	211	375	276	445	
6	101 resize	800	800	CUP	155	437	196	465	
7	101 resize	800	800	CUP	161	409	197	435	
8	102 resize	800	800	CUP	19	273	60	302	
9	102 resize	800	800	CUP	16	248	55	277	
10	102 resize	800	800	JUG	80	213	158	296	

Figure 5.9 Data content inside a CSV file

The xml data seen above is all combined and obtained as a .csv file .This file is used to generate the TFRecord file .If you see in this the 100 image number is having 2 objects in the picture . So it has 2 rows filled up with the coordinates with the class name specified that is the CUP and JUG.

5.2.4.2. TFRecord:

As we are working with large datasets, using a binary file format for storage of our data can have a significant impact on the performance of our import pipeline and for the training time of our model. Binary data takes up less space on the disk, takes less time to copy and can be read much more efficiently from disk.

Along with its advantages in performance It also makes it easy to combine multiple datasets and integrates with the data import and preprocessing functionality provided by the library. Especially for datasets that are too large to be stored fully in memory this is an advantage as only the data that is required at the time is loaded from the disk and then processed and it is possible to store sequence data.

5.2.4.2.1. Structuring TFRecords:

A TFRecord [1] file stores data as a sequence of binary strings. This means you need to specify the structure of our data before we write it to the file. Using TensorFlow TFRecords is a convenient way to get our data into our machine learning pipeline. A protocol buffer is a method developed by Google to serialize structure data in an efficient way. To generate the TFRecord files by issuing these commands from the \object detection folder

```
(tensorflow1) C:\tensorflow1\models\research\object_detection>python generate_tfrecord.py --csv_input=images\train_labels.csv --image_dir=images\train --output_path=train.record  
Successfully created the TFRecords: C:\tensorflow1\models\research\object_detection\train.record  
  
(tensorflow1) C:\tensorflow1\models\research\object_detection>python generate_tfrecord.py --csv_input=images\test_labels.csv --image_dir=images\test --output_path=test.record  
Successfully created the TFRecords: C:\tensorflow1\models\research\object_detection\test.record
```

Figure 5.10 Generation of .record file



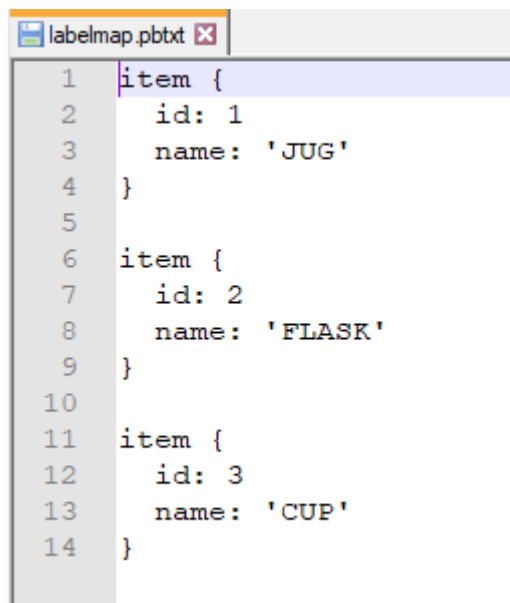
Figure 5.11 Generated .record files

The test.record and train.record file has been generated using the above python script. We are just displaying the output and the command to run the python script as some of the scripts are created on our own.

5.2.5. Create Label Map and configure Training

5.2.5.1. Label map

The label map tells the trainer what each object is by defining a mapping of class names to class ID numbers. Use a text editor we created a new file and saved it as labelmap.pbtxt.



```
1 item {
2     id: 1
3     name: 'JUG'
4 }
5
6 item {
7     id: 2
8     name: 'FLASK'
9 }
10
11 item {
12     id: 3
13     name: 'CUP'
14 }
```

Figure 5.12 Creating label map

In this class names are JUG, CUP, FLASK these names are mapped to class ID numbers. This label map ID numbers is already specified in the generate_TFRecord.py script. So this ID numbers in the labelmap.pbtxt should match with that generate_TFRecord.py

5.2.5.2. Configure Training

The object detection training pipeline must be configured. It defines which model and what parameters will be used for training. The `ssd_mobilenet_v2_coco.config` file is copied and pasted in the `\object detection\training` directory from the `\object detection\samples\configs` directory.

There are several changes to make to the `.config` file mainly changing the number of classes and examples and adding the file paths to the training data. Made following changes to the `ssd_mobilenet_v2_coco.config` file. The paths must be entered with single forward slashes, or TensorFlow will give a file path error when trying to train the model.

- Line 9 – changed `num_classes` to the number of different objects wanted to classify, it would be `num_classes: 3`.

```
model {  
  ssd {  
    num_classes: 3  
    box_coder {
```

- Line 156 – changed `file_tune_checkpoint` to:

`Fine_tune_checkpoint: "c:/tensorflow1/models/research/object_detection/ssd_mobilenet_v2_coco_2018_03_29/model.ckpt"`

```
156 fine_tune_checkpoint: "C:/tensorflow1/models/research/object_detection/ssd_mobilenet_v2_coco_2018_03_29/model.ckpt"  
157 fine_tune_checkpoint_type: "detection"
```

- Line 175 and 177 – In the train_input_reader section we changed input_path and label_map_path to :

input_path: "C:/tensorflow1/models/research/object_detection/train.record"

label_map_path:

"C:/tensorflow1/models/research/object_detection/training/labelmap.pbtxt"

```
train_input_reader: {  
  tf_record_input_reader {  
    input_path: "C:/tensorflow1/models/research/object_detection/train.record"  
  }  
  label_map_path: "C:/tensorflow1/models/research/object_detection/training/labelmap.pbtxt"  
}
```

- Line 181 – changed num_examples to the number of images having in the \images\test directory

num_examples:41

We have 41 images in the test directory that is mentioned in the code.

```
eval_config: {  
  num_examples:41
```

- Line 189 and 191 – In the eval_input_reader section, changed input_path and label_map_path to:

input_path: "C:/tensorflow1/models/research/object_detection/test.record"

label_map_path:

"C:/tensorflow1/models/research/object_detection/training/labelmap.pbtxt"

```
eval_input_reader: {  
  tf_record_input_reader {  
    input_path: "C:/tensorflow1/models/research/object_detection/test.record"  
  }  
  label_map_path: "C:/tensorflow1/models/research/object_detection/training/labelmap.pbtxt"
```

5.2.6. Run the Training:

This is the training section where we will be giving training to the pretrained model using our data. TensorFlow will initialize the training, if everything has been set up correctly. We struggled in this training part for more than a week and finally we were able to run the training section.

The command we used to begin the training:

Python train.py --logtostderr --train_dir=training/ --pipeline_config_path=training/ssd_mobilenet_v2_coco.config

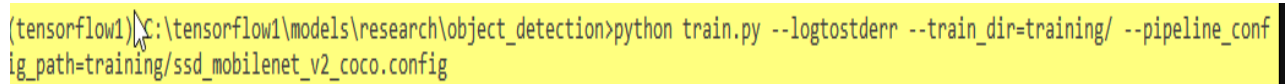
A screenshot of a terminal window with a yellow background. The command being executed is: (tensorflow1) C:\tensorflow1\models\research\object_detection>python train.py --logtostderr --train_dir=training/ --pipeline_config_path=training/ssd_mobilenet_v2_coco.config

Figure 5.13 Training command

Each steps of training reports the loss. It will start high and get lower as training progresses. For our training on the ssd_mobilenet_v2_coco model, it started at about 20 and reduced slowly. It is recommended to train until the loss consistently drops below 2.0 but it will take about 40,000 steps (depending on how powerful our CPU are) In this we have shown the loss at the step 265 and it goes on. As we are using CPU for processing so the time for each step is approximately 30 seconds. Suppose if we use GPU then the time will reduce to minimum 5 sec/step. We scripted the training routine periodically to save checkpoints about every five minutes. We can terminate the training by pressing ctrl+c. We typically wait until just after a checkpoint has been saved to terminate the training.

We can terminate training and start it later, and it will restart from the last saved checkpoint. The checkpoint at the highest number of steps will be used to generate the frozen inference graph.

```
Anaconda Prompt (anaconda3) - python train.py --logtostderr --train_dir=training/ --pipeline_config_path=training/ssd_mobilenet_v2_coco.config
INFO:tensorflow:global step 265: loss = 3.7494 (29.602 sec/step)
INFO:tensorflow:global step 266: loss = 4.0545 (35.638 sec/step)
INFO:tensorflow:global step 266: loss = 4.0545 (35.638 sec/step)
INFO:tensorflow:Recording summary at step 266.
INFO:tensorflow:Recording summary at step 266.
INFO:tensorflow:global step 267: loss = 4.0939 (34.923 sec/step)
INFO:tensorflow:global step 267: loss = 4.0939 (34.923 sec/step)
INFO:tensorflow:global step 268: loss = 3.5923 (29.821 sec/step)
INFO:tensorflow:global step 268: loss = 3.5923 (29.821 sec/step)
INFO:tensorflow:global step 269: loss = 3.9035 (32.441 sec/step)
INFO:tensorflow:global step 269: loss = 3.9035 (32.441 sec/step)
INFO:tensorflow:global step 270: loss = 4.3183 (37.475 sec/step)
INFO:tensorflow:global step 270: loss = 4.3183 (37.475 sec/step)
INFO:tensorflow:Recording summary at step 270.
INFO:tensorflow:Recording summary at step 270.
INFO:tensorflow:global step 271: loss = 4.2480 (31.547 sec/step)
INFO:tensorflow:global step 271: loss = 4.2480 (31.547 sec/step)
INFO:tensorflow:global step 272: loss = 3.8142 (29.896 sec/step)
INFO:tensorflow:global step 272: loss = 3.8142 (29.896 sec/step)
INFO:tensorflow:global step 273: loss = 4.3084 (29.765 sec/step)
INFO:tensorflow:global step 273: loss = 4.3084 (29.765 sec/step)
INFO:tensorflow:Saving checkpoint to path training/model.ckpt
INFO:tensorflow:Saving checkpoint to path training/model.ckpt
INFO:tensorflow:Recording summary at step 273.
INFO:tensorflow:Recording summary at step 273.
INFO:tensorflow:global step 274: loss = 3.7089 (38.027 sec/step)
INFO:tensorflow:global step 274: loss = 3.7089 (38.027 sec/step)
INFO:tensorflow:global step 275: loss = 4.0792 (31.448 sec/step)
INFO:tensorflow:global step 275: loss = 4.0792 (31.448 sec/step)
INFO:tensorflow:global step 276: loss = 4.1263 (29.329 sec/step)
INFO:tensorflow:global step 276: loss = 4.1263 (29.329 sec/step)
INFO:tensorflow:global step 277: loss = 3.5706 (30.750 sec/step)
INFO:tensorflow:global step 277: loss = 3.5706 (30.750 sec/step)
INFO:tensorflow:Recording summary at step 277.
INFO:tensorflow:Recording summary at step 277.
INFO:tensorflow:global step 278: loss = 3.9109 (38.592 sec/step)
INFO:tensorflow:global step 278: loss = 3.9109 (38.592 sec/step)
INFO:tensorflow:global step 279: loss = 3.6476 (32.613 sec/step)
INFO:tensorflow:global step 279: loss = 3.6476 (32.613 sec/step)
INFO:tensorflow:global step 280: loss = 3.7134 (31.374 sec/step)
INFO:tensorflow:global step 280: loss = 3.7134 (31.374 sec/step)
INFO:tensorflow:global step 281: loss = 3.7985 (38.188 sec/step)
INFO:tensorflow:global step 281: loss = 3.7985 (38.188 sec/step)
INFO:tensorflow:Recording summary at step 281.
INFO:tensorflow:Recording summary at step 281.
INFO:tensorflow:global step 282: loss = 3.8247 (32.171 sec/step)
INFO:tensorflow:global step 282: loss = 3.8247 (32.171 sec/step)
INFO:tensorflow:global step 283: loss = 3.7438 (36.641 sec/step)
INFO:tensorflow:global step 283: loss = 3.7438 (36.641 sec/step)
INFO:tensorflow:global step 284: loss = 3.8407 (39.753 sec/step)
INFO:tensorflow:global step 284: loss = 3.8407 (39.753 sec/step)
INFO:tensorflow:Recording summary at step 284.
INFO:tensorflow:Recording summary at step 284.
INFO:tensorflow:global step 285: loss = 3.8127 (39.228 sec/step)
INFO:tensorflow:global step 285: loss = 3.8127 (39.228 sec/step)
INFO:tensorflow:global step 286: loss = 3.4020 (34.406 sec/step)
INFO:tensorflow:global step 286: loss = 3.4020 (34.406 sec/step)
INFO:tensorflow:global step 287: loss = 4.0698 (30.741 sec/step)
INFO:tensorflow:global step 287: loss = 4.0698 (30.741 sec/step)
INFO:tensorflow:Recording summary at step 287.
INFO:tensorflow:Recording summary at step 287.
INFO:tensorflow:global step 288: loss = 3.1382 (38.290 sec/step)
INFO:tensorflow:global step 288: loss = 3.1382 (38.290 sec/step)
```

Figure 5.14 Training process

We can view the progress of the training job by using tensor board. To do this, open a new instance of Anaconda prompt, activate the tensorflow1 virtual environment, and change to the object detection directory and issue following command:

```
(tensorflow1) C:\tensorflow1\models\research\object_detection>tensorboard --logdir=training --host 192.168.1.5
```

Figure 5.15 Command for Tensorboard

This will create a webpage on our local host which can be viewed through a web browser. The Tensor board page provides information and graphs that show how the training is progressing. One important graph is the loss graph, which shows the loss of the classifier over time

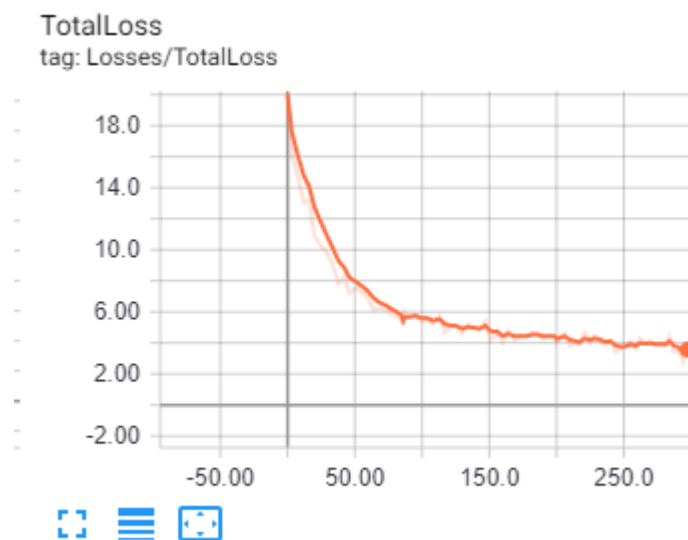


Figure 5.16 Total loss graph

From this we can understand that at first the loss starts from above 18 which is exactly 20 as already mentioned. The x-axis which represents the number of steps as we have 300 steps the it is shown in the graph. The y-axis represents the loss value. At first the loss was 20 for 1st step then after each step increasing the loss reduces rapidly and it comes to around 3.45 loss.

5.2.7. Export inference graph:

When we need to keep all the values of the variables and the Graph in a single file, we do it with freezing the graphs. To reduce the amount of computation needed when the network is used only for inferences, we can remove some parts of a graph that are only needed for training.

For example

- Removing operations used only for training like checkpoint saving.
- Stripping out parts of the graph that are never reached.
- Removing debug operations like CheckNumerics.

Now the training is complete, next is to generate the frozen inference graph (.pb file)

```
(tensorflow1) C:\tensorflow1\models\research\object_detection>python export_inference_graph.py --input_type image_tensor --pipeline_config_path training/ssd_mobilenet_v2_coco.config --trained_checkpoint_prefix training/model.ckpt-300 --output_directory inference_graph
```

Figure 5.17 Command used for generating inference graph

PC > Local Disk (C:) > tensorflow1 > models > research > object_detection > inference_graph			
Name	Date modified	Type	Size
saved_model	04-May-20 7:07 PM	File folder	
checkpoint	04-May-20 7:07 PM	File	1 KB
frozen_inference_graph.pb	04-May-20 7:07 PM	PB File	18,826 KB
model.ckpt.data-00000-of-00001	04-May-20 7:07 PM	DATA-00000-OF-0...	18,287 KB
model.ckpt.index	04-May-20 7:07 PM	INDEX File	14 KB
model.ckpt.meta	04-May-20 7:07 PM	META File	1,294 KB
pipeline	04-May-20 7:07 PM	XML Configuratio...	5 KB

Figure 5.18 Files inside Inference graph folder

After saving the models. There will be four files:

- **model.ckpt.meta**: TensorFlow stores the graph structure separately from the variable values. The file *.ckpt.meta* contains the complete graph. It includes GraphDef, SaverDef and so on
- **model.ckpt.index**: It is a table where each key is the name of a tensor and its value is a serialized bundleEntryProto
- **model.ckpt.data-00000-of-00001**: This contains the values of variables (weights, biases, placeholders, gradients, hyper-parameters etc.).

NOTE: Serialized BundleEntryProto holds metadata of the tensors. Metadata of a tensor may be like: which of the “data” files contains the content of a tensor, the offset into that file, checksum, some auxiliary data, etc.

- **checkpoint**: All checkpoint information, like model *ckpt file* name and path.

This creates a frozen_inference_graph.pb file in the \object_detection\inference_graph folder. The .pbfile contains the object detection classifier

5.3. TensorFlow Lite Object Detection

5.3.1. Introduction

TensorFlow Lite is an optimized framework for deploying lightweight deep learning models on resource-constrained edge devices. TensorFlow Lite models have faster inference time and require less processing power, so they can be used to obtain faster performance in real-time applications. We trained a custom TensorFlow Object Detection model, so now we convert it into an optimized format that can be used by TensorFlow Lite and run it on Raspberry pi.

5.3.1.1. Features

- **Lightweight** –

We have heard of mobile models like mobile nets and how they are designed for the mobile platform. They are Lightweight working on small low power devices like raspberry pi, phones

- **Low Latency** –

TensorFlow is a designed to run on devices with low latency and without the need for an internet connection

- **Privacy** –

Since TFLite uses on-device ML to operate, there's absolutely no need for data to leave the device in sharing our privacy.

- **Low power consumption** –

It improves power consumption as that network connections can tend to be very power-hungry.

- **Efficient model format** –

Models in TFLite are designed to have a small binary size with just a minor impact on accuracy.

- **Pre trained models** –

TensorFlow Lite has just about everything you need for the most common machine learning tasks.

5.3.1.2. Components

TFLite consists of two components.

Table 5.2 Components in TensorFlow Lite

Converter (to TensorFlow Lite format)	Interpreter (core)
Transforms TensorFlow models in to a form efficient for reading by the interpreter	Diverse platform support Android, IOS, embedded Linux and microcontrollers
Introduces optimizations to improve binary size model performance and reduce model size	Platform APIs for accelerated inference

5.3.1.3. Architecture

Once we have trained model, it has to go through a process of conversion before it can be used on a device first, we have to train your model with TensorFlow. And typically, you'll then save it out in a common format like the recommended saved model format. Once we have it we will use the TensorFlow light converter tools to flatten the model to prepare it for mobile or embedded devices. Running inference on compute heavy machine learning models on mobile devices is resource demanding due to a device has limited processing and power. So inference on these devices has to be performed very quickly to avoid overhead and make real-time applications possible for this purpose TensorFlow Lite can employ Hardware acceleration libraries or APIs for supported devices.

5.3.2. Export Frozen Inference graph for TensorFlow Lite:

Now the training has finished, the model can be exported for conversion to TensorFlow Lite using the `export_tflite_ssd_graph.py` script. For this first we created a folder in `\object_detection` called “TFLite_model”. Next, to set up some environment variables so the commands are easier to type out.

```
(tensorflow1) C:\tensorflow1\models\research\object_detection>set CONFIG_FILE=C:\\tensorflow1\models\research\object_detection\training\ssd_mobilenet_v2_coco.config
(tensorflow1) C:\tensorflow1\models\research\object_detection>set CHECKPOINT_PATH=C:\\tensorflow1\models\research\object_detection\training\model.ckpt-300
(tensorflow1) C:\tensorflow1\models\research\object_detection>set OUTPUT_DIR=C:\\tensorflow1\models\research\object_detection\TFLite_model
```

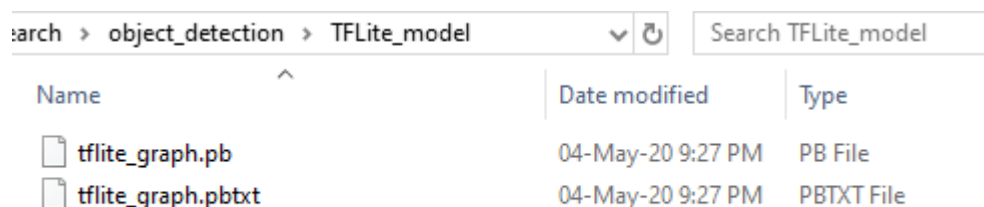
Figure 5.19 Setting up environment variables

Now that those are set up, issue this command to export the model for TensorFlow Lite:

```
(tensorflow1) C:\tensorflow1\models\research\object_detection>python export_tflite_ssd_graph.py --pipeline_config_path=%CONFIG_FILE% --trained_checkpoint_prefix=%CHECKPOINT_PATH% --output_directory=%OUTPUT_DIR% --add_postprocessing_op=true
```

Figure 5.20 Command for exporting TFLite model

After the command has executed, there should be two new files in the `\object_detection\TFLite_model` folder: `tflite_graph.pb` and `tflite_graph.pbtxt`.





search > object_detection > TFLite_model			Search TFLite_model
Name	Date modified	Type	
 tflite_graph.pb	04-May-20 9:27 PM	PB File	
 tflite_graph.pbtxt	04-May-20 9:27 PM	PBTXT File	

Figure 5.21 Files inside TFLite folder

The new inference graph has been trained and exported. This inference graph's architecture and network operations are compatible with TFLite's framework. Now the graph will be converted to an actual TensorFlow Lite model. After this conversion we will use it in the raspberry pi for our project.

5.3.3. CREATING OPTIMIZED TENSORFLOW LITE MODEL:

As we have already exported a frozen graph of our detection model for TensorFlow Lite, we still need to run it through the TensorFlow Lite Optimizing Converter (TOCO) before it will work with the TensorFlow Lite interpreter. TOCO converts models in an optimized Flat Buffer format that allows them to run efficiently on TensorFlow Lite. We also need to create a new label map before running the model. Thus this TOCO will convert the resulting frozen graph (tflite_graph.pb) to the TensorFlow Lite flat buffer format (detect.tflite) via the following command. For floating model, run this command.

5.3.3.1. Algorithm:

This command takes our input tensor `normalized_input_image_tensor` after resizing each camera image frame to 300x300 pixels. We will be specifying the output directory to save the `detect.tflite` output. The `graph_def_file` command is that where we will be providing the input file `tflite_graph.pb` directory. The outputs of the model are named `'TFLite_Detection_PostProcess','TFLite_Detection_PostProcess:1',`
`'TFLite_Detection_PostProcess:2'` and `'TFLite_Detection_PostProcess:3'`

5.3.3.2. TFLite command:

```
tflite_convert --output_file=/home/arun/Downloads/detect.tflite --  
graph_def_file=/home/arun/Downloads/tflite_graph.pb --input_shapes=1,300,300,3 -
```

```
-input_arrays=normalized_input_image_tensor --  
output_arrays='TFLite_Detection_PostProcess','TFLite_Detection_PostProcess:1',  
TFLite_Detection_PostProcess:2','TFLite_Detection_PostProcess:3' --  
inference_type=FLOAT --allow_custom_ops
```

```
(venv) arun@arun-HP-Laptop-15-bs1xx:~/Downloads$ tflite_convert --output_file=/home/arun/Downloads/detect.tflite --graph_def_file=/home/arun/Downloads/tflite_graph.pb --input_shapes=1,300,300,3 --input_arrays=normalized_input_image_tensor --output_arrays='TFLite_Detection_PostProcess','TFLite_Detection_PostProcess:1','TFLite_Detection_PostProcess:2','TFLite_Detection_PostProcess:3' --inference_type=FLOAT --allow_custom_ops
```

Figure 5.22 Command used for Creating detect.tflite file

Before this command downloads folder contains only one tflite_graph.pb file. This is the file to be converted to detect.tflite file.

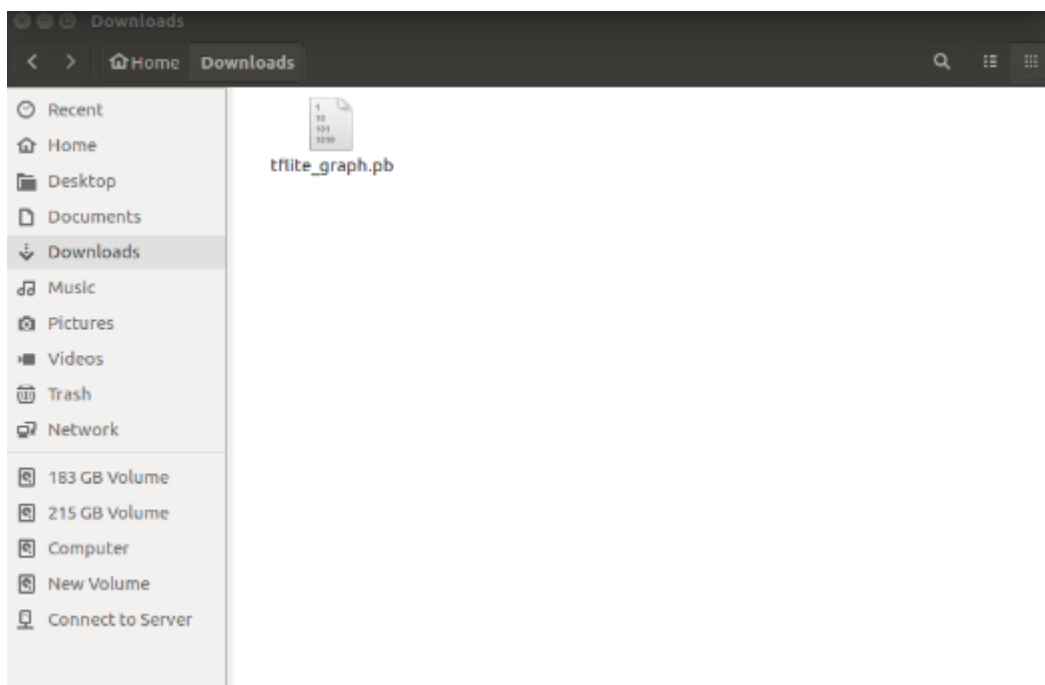


Figure 5.23 Generated TF_Lite File

```
arun@arun-HP-Laptop-15-bs1xx:~$ source venv/bin/activate  
(venv) arun@arun-HP-Laptop-15-bs1xx:~$ cd Downloads/  
(venv) arun@arun-HP-Laptop-15-bs1xx:~/Downloads$ ls  
tflite_graph.pb
```

After this command execution we will get the detect.tflite file in the Downloads\ directory which is the file to be used in raspberry pi (embedded systems), mobile devices.

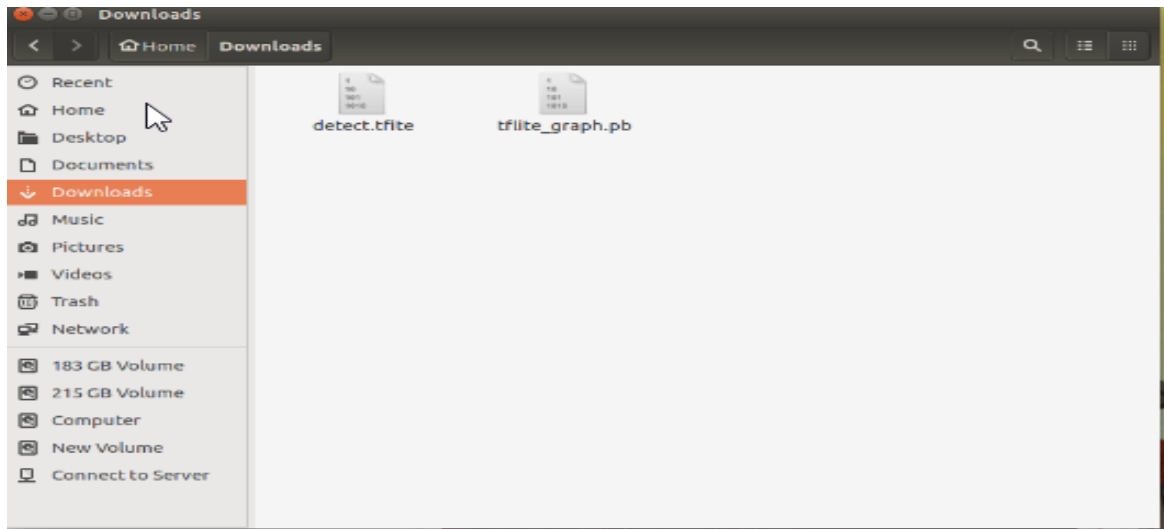


Figure 5.24 Generation of Detect.Tflite file in Raspberry Pi

5.3.4. TENSORFLOW OBJECT DETECTION API ON THE RASPBERRY PI

The following steps are done in Raspberry Pi to perform object detection API on live video feeds from a Picamera or USB webcam. This module includes the Objectdetectionpicamera.py script, which is a Python script that loads an object detection model in TensorFlow and uses it to detect objects in a Picamera video feed. We use TensorFlow v1.8.0 on a Raspberry Pi Model 3B+ running on Raspbian Stretch.

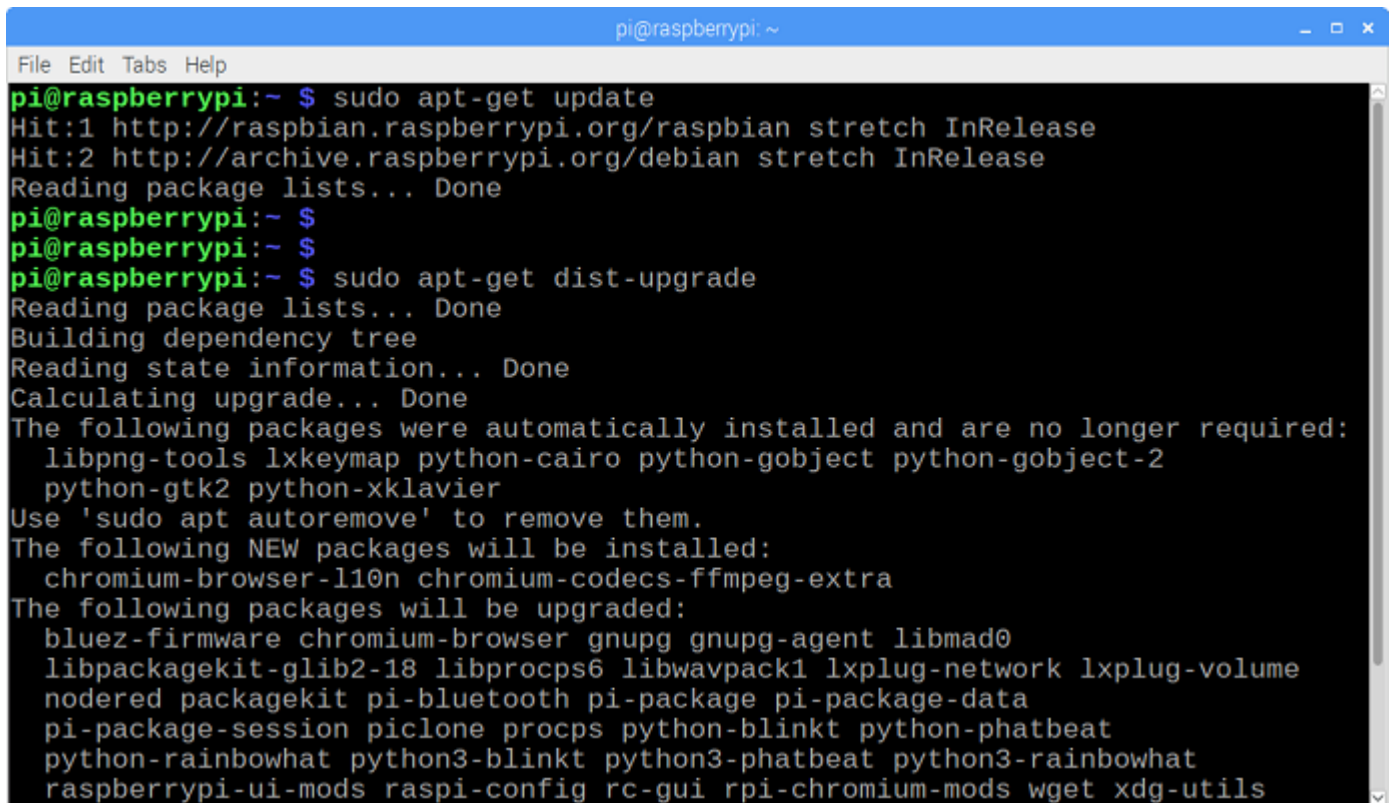
Step 1: Update the Raspberry Pi

First, the Raspberry Pi needs to be fully updated. Open the terminal window and type:

sudo apt-get update

sudo apt-get upgrade

The upgrade will happen anywhere between a minute and an hour



```
pi@raspberrypi:~ $ sudo apt-get update
Hit:1 http://raspbian.raspberrypi.org/raspbian stretch InRelease
Hit:2 http://archive.raspberrypi.org/debian stretch InRelease
Reading package lists... Done
pi@raspberrypi:~ $
pi@raspberrypi:~ $
pi@raspberrypi:~ $ sudo apt-get dist-upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages were automatically installed and are no longer required:
  libpng-tools lxkeymap python-cairo python-gobject python-gobject-2
  python-gtk2 python-xklavier
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  chromium-browser-l10n chromium-codecs-ffmpeg-extra
The following packages will be upgraded:
  bluez-firmware chromium-browser gnupg gnupg-agent libmad0
  libpackagekit-glib2-18 libprocps6 libwavpack1 lxplug-network lxplug-volume
  nodered packagekit pi-bluetooth pi-package pi-package-data
  pi-package-session piclone procps python-blinkt python-phatbeat
  python-rainbowhat python3-blinkt python3-phatbeat python3-rainbowhat
  raspberrypi-ui-mods raspi-config rc-gui rpi-chromium-mods wget xdg-utils
```

Figure 5.25 Issuing update and upgrade commands in terminal window

Step 2. Install TensorFlow

The next step is to install TensorFlow which is more than 100MB in size. The TensorFlow is installed by using the following command in the terminal window.

pip3 install TensorFlow

TensorFlow also needs some dependencies to work with so in order to install such dependencies in the terminal window we issue the following commands:

sudo pip3 install pillow lxmljupyter matplotlib cython

sudo apt-get install python-tk

Step 3. Install OpenCV

The next step is to install OpenCV by installing all its dependencies first and installing OpenCV by issuing the following commands in the terminal window:

```
sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
```

```
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
```

```
sudo apt-get install libxvidcore-dev libx264-dev
```

```
sudo apt-get install qt4-dev-tools libatlas-base-dev
```

Now that we've got all those installed, we can install OpenCV. Issue

```
sudo pip3 install opencv-python
```

Step 4. Compile and Install Protobuf

The TensorFlow object detection API uses Protobuf, a package that implements Google's Protocol Buffer data format. We need to compile this from source, by issuing the following command in terminal window:

```
sudo apt-get install protobuf-compiler
```

After installing the protoc version is made to run.

Step 5. Set up TensorFlow Directory Structure and PythonPath Variable

In the previous steps the packages are installed, the next step is to set up a TensorFlow path variable, for that we must move back to the tensor flow directory and make cd into it.

```
mkdir tensorflow1
```

```
cd tensorflow1
```

Next, we need to modify the PYTHONPATH environment variable to point at some directories inside the TensorFlow repository we just downloaded. We want PYTHONPATH to be set every time we open a terminal, so we have to modify the .bashrc file. Open it by issuing:

```
sudo nano ~/.bashrc
```

Move to the end of the file, and on the last line, add:

```
export
```

```
PYTHONPATH=$PYTHONPATH:/home/pi/tensorflow1/models/research:/home/pi/tensorflow1/models/research/slim
```

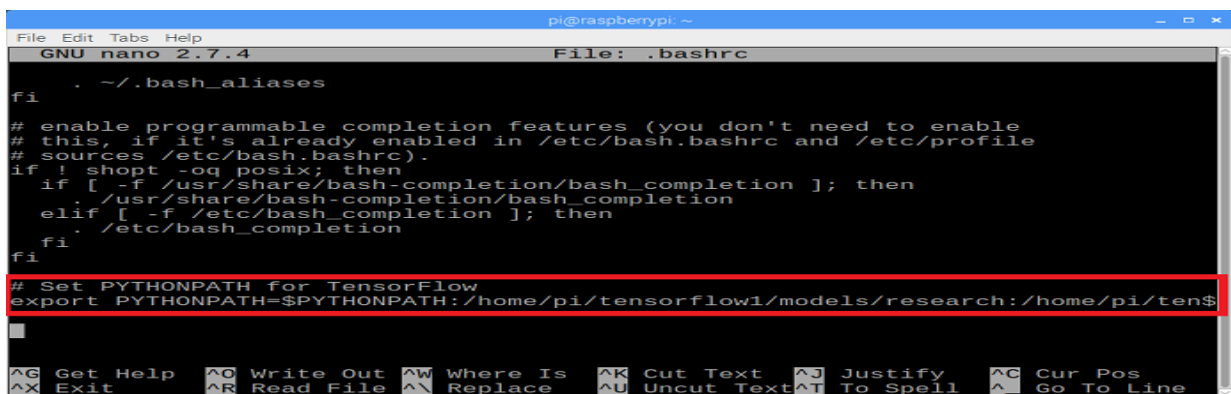


Figure 5.26 Setting pythonpath in terminal window

Then, save and exit the file. This makes it so the “export PYTHONPATH” command is called every time you open a new terminal, so the PYTHONPATH variable will always be set appropriately. Close and then re-open the terminal. Now, we need to use Protoc to compile the Protocol Buffer (. proto) files used by the Object Detection API. The . proto files are located in /research/object_detection/protos, but we need to execute the command from the /research directory.

Issue:

```
cd /home/pi/tensorflow1/models/research
```

```
protoc object_detection/protos/*.proto --python_out=.
```

This command converts all the "name".proto files to "name_pb2".py files. Next, move into the object_detection directory:

```
cd /home/pi/tensorflow1/models/research/object_detection
```

Now, we have to download the SSD Lite model from the TensorFlow detection model zoo. The model zoo is Google's collection of pre-trained object detection models that have various levels of speed and accuracy. The Raspberry Pi has a weak processor, so we need to use a model that takes less processing power. Though the model will run faster, it comes at a tradeoff of having lower accuracy. We have to download the SSD Lite MobileNet model and unpack it by issuing:

```
wget
```

```
http://download.tensorflow.org/models/object_detection/ssdlite_mobilenet_v2_coco_2018_05_09.tar.gz  
tar -xzf ssdlite_mobilenet_v2_coco_2018_05_09.tar.gz
```

Now the model is in the object_detection directory and ready to be used.

Step 6. Detect Objects

The Object_detection_picamera.py, detects objects in live feeds from a Picamera or USB webcam. Basically, the script sets paths to the model and label map, loads the model into memory, initializes the Picamera, and then begins performing object detection on each video frame from the Picamera. Before using Pi Camera, we have to enable its configuration.

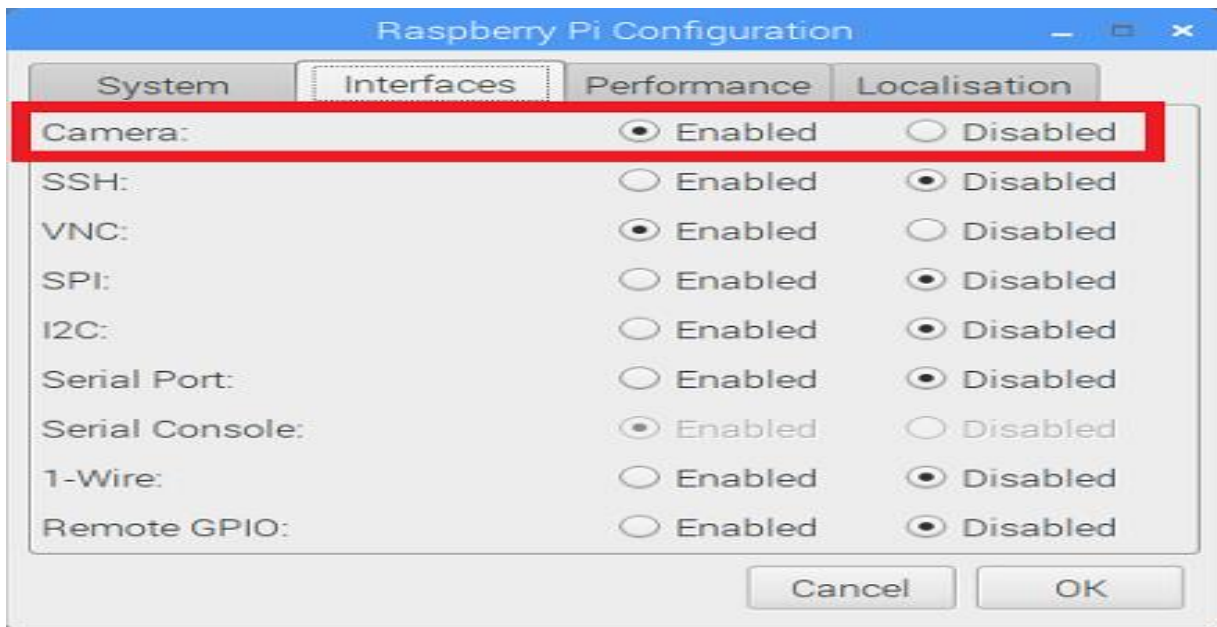


Figure 5.27 Enabling Camera Module in Raspberry Pi configuration menu

Then we have to Download the `Object_detection_picamera.py` file into the `object_detection`, the for running the pi camera we should issue:

`python3 Object_detection_picamera.py`

The script defaults to using an attached Picamera.

Once the script initializes (which can take up to 30 seconds), we will see a window showing a live view from our camera. Common objects inside the view will be identified and have a rectangle drawn around them.

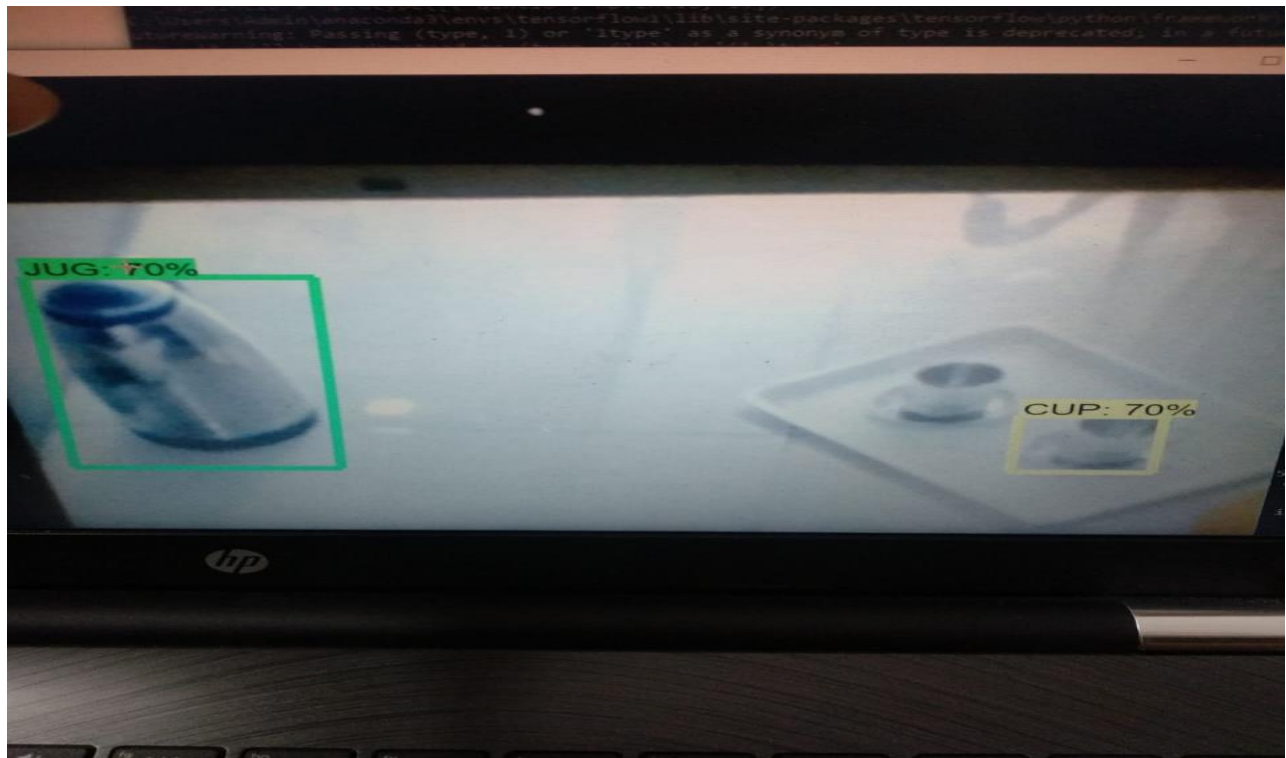


Figure 5.28 Final Output of testing

With the SSDLite model, the Raspberry Pi 3 performs fairly well, achieving a frame rate higher than 1FPS. This is fast enough for most real-time object detection applications. Now, open the `Object_detection_picamera.py` script in a text editor. Go to the line where `MODEL_NAME` is set and change the string to match the name of the new model folder. Then, on the line where `PATH_TO_LABELS` is set, change the name of the labelmap file to match the new label map. Change the `NUM_CLASSES` variable to the number of classes your model can identify. Now, when we run the script, it will use our model rather than the SSDLiteMobileNet model.

CHAPTER-6

FUTURE SCOPE AND CONCLUSION

The proposed system was trained with 3 different kinds of datasets namely jug, cup and flask using SSDmobilenet model v2, and the test results were with 96% accuracy and 0.9 frames per second. Although the accuracy seems to be good, the frames per second is very low for a real time detection. In case of low traffic object detection where the objects are slow moving through the frame, then we can certainly consider using the Raspberry Pi for deep learning object detection. However, if we are developing an application that involves many objects that are fast moving, we should instead consider faster hardware.

In future this application could be improved with the increased processing power by using a tool named Coral which is a USB device that provides an edge Tensor Processing Unit (TPU) that highly accelerates machine learning model inference when attached to a Linux host computer (including Raspberry Pi), such that frames per second could be enhanced when compared to raspberry pi which operates on a lesser RAM of 1 GB. Thus, the project will be useful to detect and track the objects and makes the life easier.

REFERENCECS

- [1] George E.Sakr ,Maria Mokbel,Ali Hadi and Ahmad Darwich ,COMPARING DEEP LEARNING AND SUPPORT VECTOR MACHINES FOR AUTONOMOUS WASTE SORTING in 2016 IEEE International Multidisciplinary Conference on Electrical Technology (IMCET) ,vol 26,no.3,pp.657-853,March 2016
- [2] Tiagarajah ,V.Janahiraman¹ and Mohammad Shahrul Subuhan ,TRAFFIC LIGHT USING TENSORFLOW OBJECT DETECTION FRAMEWORK in 2017 IEEE 9th International Conference on System Engineering and Technology (ICSET),vol 27,no.4,pp.567-598,March 2017.
- [3] Abdelhakim Zeroual ,P.F Rocha and M.P Silva,SSD MultiBox Object Detection ,published in the Proceedings of the International Conference on Networking and Advanced Systems (ICNAS) IEEE,vol 25,no.2,pp.543-538,March 2018.
- [4] Wei Liu,Dragomir Anguelov,Dumirtu Erhan,Christian Szegedy,"SSD:Single Shot MultiBox Detector " in arxiv Cornell University,2019.
- [5] De Charette, R. and Nashashibi, F., "Traffic light recognition using image processing compared to learning processes," In 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, vol 15,no.6,pp.345-214,October 2009.
- [6] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y. and Berg, A.C., "Ssd: Single shot multibox detector," in ArXiv,20 Dec 2014.
- [8] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: "Single shot multibox detector",.In European Conference on Computer Vision.
- [9] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. "SSD: Single Shot MultiBox Detector". In ArXiv , 30 Mar 2018.

[10] <https://www.raspberrypi.org/about/>

[11] Python documentation, docs.python.org

[12] <https://www.tensorflow.org/about/>

APPENDIX

Source code for training:

```
import functools
import json
import os
import tensorflow as tf

from object_detection.builders import dataset_builder
from object_detection.builders import graph_rewriter_builder
from object_detection.builders import model_builder
from object_detection.legacy import trainer
from object_detection.utils import config_util

tf.logging.set_verbosity(tf.logging.INFO)

flags = tf.app.flags
flags.DEFINE_string('master', '', 'Name of the TensorFlow master to use.')
flags.DEFINE_integer('task', 0, 'task id')
flags.DEFINE_integer('num_clones', 1, 'Number of clones to deploy per worker.')
flags.DEFINE_boolean('clone_on_cpu', False,
                    'Force clones to be deployed on CPU. Note that even if '
                    'set to False (allowing ops to run on gpu), some ops may '
                    'still be run on the CPU if they have no GPU kernel.')
flags.DEFINE_integer('worker_replicas', 1, 'Number of worker+trainer'
                    'replicas.')
```

```

flags.DEFINE_integer ('ps_tasks', 0,
                      'Number of parameter server tasks. If None, does not use '
                      'a parameter server.')
flags.DEFINE_string ('train_dir', '',
                    'Directory to save the checkpoints and training summaries.')

flags.DEFINE_string ('pipeline_config_path', '',
                    'Path to a pipeline_pb2.TrainEvalPipelineConfig config '
                    'file. If provided, other configs are ignored')

flags.DEFINE_string ('train_config_path', '',
                    'Path to a train_pb2.TrainConfig config file.')
flags.DEFINE_string ('input_config_path', '',
                    'Path to an input_reader_pb2.InputReader config file.')
flags.DEFINE_string ('model_config_path', '',
                    'Path to a model_pb2.DetectionModel config file.')

FLAGS = flags.FLAGS

@tf.contrib.framework.deprecated (None, 'Use
object_detection/model_main.py.')
def main(_):
    assert FLAGS.train_dir, '`train_dir` is missing.'
    if FLAGS.task == 0: tf.gfile.MakeDirs(FLAGS.train_dir)
    if FLAGS.pipeline_config_path:
        configs = config_util.get_configs_from_pipeline_file(
            FLAGS.pipeline_config_path)
        if FLAGS.task == 0:
            tf.gfile.Copy(FLAGS.pipeline_config_path,
                os.path.join(FLAGS.train_dir, 'pipeline.config'),

```

```

        overwrite=True)

    else:
        configs = config_util.get_configs_from_multiple_files(
            model_config_path=FLAGS.model_config_path,
            train_config_path=FLAGS.train_config_path,
            train_input_config_path=FLAGS.input_config_path)
        if FLAGS.task == 0:
            for name, config in [('model.config', FLAGS.model_config_path),
                                ('train.config', FLAGS.train_config_path),
                                ('input.config', FLAGS.input_config_path)]:
                tf.gfile.Copy(config, os.path.join(FLAGS.train_dir, name),
                              overwrite=True)

    model_config = configs['model']
    train_config = configs['train_config']
    input_config = configs['train_input_config']

    model_fn = functools.partial(
        model_builder.build,
        model_config=model_config,
        is_training=True)

    def get_next(config):
        return dataset_builder.make_initializable_iterator(
            dataset_builder.build(config)).get_next()

    create_input_dict_fn = functools.partial(get_next, input_config)

    env = json.loads(os.environ.get('TF_CONFIG', '{}'))

```

```

cluster_data = env.get('cluster', None)
cluster = tf.train.ClusterSpec(cluster_data) if cluster_data else None
task_data = env.get('task', None) or {'type': 'master', 'index': 0}
task_info = type('TaskSpec', (object,), task_data)

# Parameters for a single worker.
ps_tasks = 0
worker_replicas = 1
worker_job_name = 'lonely_worker'
task = 0
is_chief = True
master = ""

if cluster_data and 'worker' in cluster_data:
    # Number of total worker replicas include "worker"s and the "master".
    worker_replicas = len(cluster_data['worker']) + 1
    if cluster_data and 'ps' in cluster_data:
        ps_tasks = len(cluster_data['ps'])

if worker_replicas > 1 and ps_tasks < 1:
    raise ValueError('At least 1 ps task is needed for distributed training.')

if worker_replicas >= 1 and ps_tasks > 0:
    # Set up distributed training.
    server = tf.train.Server(tf.train.ClusterSpec(cluster), protocol='grpc',
job_name=task_info.type,
task_index=task_info.index)
    if task_info.type == 'ps':
server.join()

```

```

    return

worker_job_name = '%s/task:%d' % (task_info.type, task_info.index)
    task = task_info.index
is_chief = (task_info.type == 'master')
    master = server.target

graph_rewriter_fn = None
    if 'graph_rewriter_config' in configs:
graph_rewriter_fn = graph_rewriter_builder.build(
    configs['graph_rewriter_config'], is_training=True)

trainer.train(
    create_input_dict_fn,
    model_fn,
    train_config,
        master,
        task,
    FLAGS.num_clones,
    worker_replicas,
    FLAGS.clone_on_cpu,
    ps_tasks,
    worker_job_name,
    is_chief,
    FLAGS.train_dir,
    graph_hook_fn=graph_rewriter_fn)

if __name__ == '__main__':

```

```
tf.app.run()
```

SOURCE CODE FOR OBJECT DETECTION:

```
import os
import cv2
import numpy as np
from picamera.array import PiRGBArray
from picamera import PiCamera
import tensorflow as tf
import argparse
import sys

# Set up camera constants
IM_WIDTH = 1280
IM_HEIGHT = 720
#IM_WIDTH = 640    Use smaller resolution for
#IM_HEIGHT = 480   slightly faster framerate

# Select camera type (if user enters --usbcam when calling this script,
# a USB webcam will be used)
camera_type = 'picamera'
parser = argparse.ArgumentParser()
parser.add_argument('--usbcam', help='Use a USB webcam instead of
picamera',
                    action='store_true')
args = parser.parse_args()
if args.usbcam:
```

```
camera_type = 'usb'

# This is needed since the working directory is the object_detection folder.
sys.path.append('..')

# Import utilites
from utils import label_map_util
from utils import visualization_utils as vis_util

# Name of the directory containing the object detection module we're using
MODEL_NAME = 'ssdlite_mobilenet_v2_coco_2018_05_09'

# Grab path to current working directory
CWD_PATH = os.getcwd()

# Path to frozen detection graph .pb file, which contains the model that is used
# for object detection.
PATH_TO_CKPT =
os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference_graph.pb')

# Path to label map file
PATH_TO_LABELS =
os.path.join(CWD_PATH,'data','mascoco_label_map.pbtxt')

# Number of classes the object detector can identify
NUM_CLASSES = 90

## Load the label map.
# Label maps map indices to category names, so that when the convolution
```



```

# network predicts `5`, we know that this corresponds to `airplane`.
# Here we use internal utility functions, but anything that returns a
# dictionary mapping integers to appropriate string labels would be fine
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)

# Load the Tensorflow model into memory.
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.GraphDef()
    with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name="")

sess = tf.Session(graph=detection_graph)

# Define input and output tensors (i.e. data) for the object detection classifier

# Input tensor is the image
image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')

# Output tensors are the detection boxes, scores, and classes
# Each box represents a part of the image where a particular object was detected
detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')

```

```

# Each score represents level of confidence for each of the objects.
# The score is shown on the result image, together with the class label.
detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
detection_classes = detection_graph.get_tensor_by_name('detection_classes:0')

# Number of objects detected
num_detections = detection_graph.get_tensor_by_name('num_detections:0')

# Initialize frame rate calculation
frame_rate_calc = 1
freq = cv2.getTickFrequency()
font = cv2.FONT_HERSHEY_SIMPLEX

# Initialize camera and perform object detection.
# The camera has to be set up and used differently depending on if it's a
# Picamera or USB webcam.

# I know this is ugly, but I basically copy+pasted the code for the object
# detection loop twice, and made one work for Picamera and the other work
# for USB.

#### Picamera ####
if camera_type == 'picamera':
    # Initialize Picamera and grab reference to the raw capture
    camera = PiCamera()
    camera.resolution = (IM_WIDTH,IM_HEIGHT)
    camera.framerate = 10
    rawCapture = PiRGBArray(camera, size=(IM_WIDTH,IM_HEIGHT))
    rawCapture.truncate(0)

```

```

for frame1 in camera.capture_continuous(rawCapture,
format="bgr",use_video_port=True):

    t1 = cv2.getTickCount()

    # Acquire frame and expand frame dimensions to have shape: [1, None,
None, 3]
    # i.e. a single-column array, where each item in the column has the pixel
RGB value
    frame = np.copy(frame1.array)
    frame.setflags(write=1)
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    frame_expanded = np.expand_dims(frame_rgb, axis=0)

    # Perform the actual detection by running the model with the image as
input
    (boxes, scores, classes, num) = sess.run(
        [detection_boxes, detection_scores, detection_classes, num_detections],
        feed_dict={image_tensor: frame_expanded})

    # Draw the results of the detection (aka 'visulaize the results')
    vis_util.visualize_boxes_and_labels_on_image_array(
        frame,
        np.squeeze(boxes),
        np.squeeze(classes).astype(np.int32),
        np.squeeze(scores),
        category_index,
        use_normalized_coordinates=True,

```

```

line_thickness=8,
min_score_thresh=0.40)

    cv2.putText(frame,"FPS:
{0:.2f}".format(frame_rate_calc),(30,50),font,1,(255,255,0),2,cv2.LINE_AA)

    # All the results have been drawn on the frame, so it's time to display it.
    cv2.imshow('Object detector', frame)

    t2 = cv2.getTickCount()
    time1 = (t2-t1)/freq
    frame_rate_calc = 1/time1

    # Press 'q' to quit
    if cv2.waitKey(1) == ord('q'):
        break

rawCapture.truncate(0)

camera.close()

### USB webcam ###
elif camera_type == 'usb':
    # Initialize USB webcam feed
    camera = cv2.VideoCapture(0)
    ret = camera.set(3,IM_WIDTH)
    ret = camera.set(4,IM_HEIGHT)

while(True):

```

```

t1 = cv2.getTickCount()

# Acquire frame and expand frame dimensions to have shape: [1, None,
None, 3]
# i.e. a single-column array, where each item in the column has the pixel
RGB value
ret, frame = camera.read()
frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
frame_expanded = np.expand_dims(frame_rgb, axis=0)

# Perform the actual detection by running the model with the image as
input
(boxes, scores, classes, num) = sess.run(
    [detection_boxes, detection_scores, detection_classes, num_detections],
    feed_dict={image_tensor: frame_expanded})

# Draw the results of the detection (aka 'visulaize the results')
vis_util.visualize_boxes_and_labels_on_image_array(
    frame,
    np.squeeze(boxes),
    np.squeeze(classes).astype(np.int32),
    np.squeeze(scores),
    category_index,
    use_normalized_coordinates=True,
    line_thickness=8,
    min_score_thresh=0.85)

```

```
cv2.putText(frame,"FPS:
{0:.2f}".format(frame_rate_calc),(30,50),font,1,(255,255,0),2,cv2.LINE_AA)
```

```
# All the results have been drawn on the frame, so it's time to display it.
```

```
cv2.imshow('Object detector', frame)
```

```
t2 = cv2.getTickCount()
```

```
time1 = (t2-t1)/freq
```

```
frame_rate_calc = 1/time1
```

```
# Press 'q' to quit
```

```
if cv2.waitKey(1) == ord('q'):
```

```
    break
```

```
camera.release()
```

```
cv2.destroyAllWindows()
```